VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

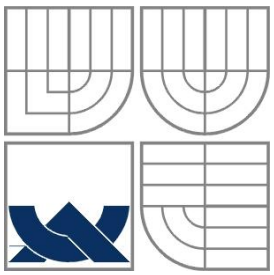
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOMÉNOVÉ INDEXY V PROSTŘEDÍ ORACLE 11G

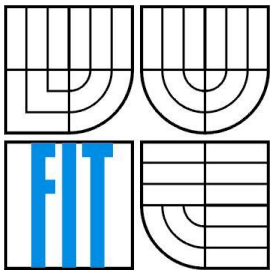
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAN DVOŘÁK
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOMÉNOVÉ INDEXY V PROSTŘEDÍ ORACLE 11G

DOMAIN INDICES IN ORACLE 11G

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE — Bc. JAN DVOŘÁK
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2011

Abstrakt

Diplomová práce se zabývá problematikou doménových indexů v Oracle Database 11g. Popisuje architekturu databáze a rozebírá dostupné možnosti indexování. Jsou zde vysvětleny konkrétní způsoby implementace a použití doménových indexů, dále rozebírány způsoby indexování časoprostorových dat, především struktura TB-stromu, která je následně implementována v podobě doménového indexu. Spolu s doménovým indexem jsou implementovány také operátory, pomocí kterých je index následně využíván a testován.

Abstract

This thesis deals with the domain indexes in Oracle Database 11g. It describes the database architecture and discusses the available methods of indexing. There are explained concrete ways of the implementation and use of domain indexes, also discussed ways of indexing spatio-temporal data especially the TB-tree structure, which is then implemented as a domain index. Along with the domain index operators are also implemented by means of which the index is subsequently used and tested.

Klíčová slova

Oracle 11g, rozšířené indexování, doménový index, indexový typ, ODCI, TB-strom, časoprostorová data, PL/SQL

Keywords

Oracle 11g, extensible indexing, domain index, indextype, ODCI, TB-tree, spatio-temporal data, PL/SQL

Citace

Dvořák Jan: Doménové indexy v prostředí Oracle 11g, diplomová práce, Brno, FIT VUT v Brně, 2011

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Doc. Ing. Jaroslava Zendulky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Dvořák
14. května 2011

Poděkování

Na tomto místě bych chtěl poděkovat panu Doc. Ing. Jaroslavu Zendulkovi, CSc. za jeho pomoc a vstřícnost při řešení této diplomové práce.

Obsah

1 Úvod

V poslední době dramaticky narůstá objem komplexních typů dat, jako jsou například prostorová nebo multimediální data, dokumenty, obrázky, záznamy DNA apod. Přestože tato data nejsou databázemi nativně zpracovávána, je třeba je někam ukládat a následně s nimi i efektivním způsobem pracovat. Právě z tohoto důvodu systém Oracle podporuje indexování komplexních datových typů pomocí rozšířeného indexování (extensible indexing), při kterém jsou vytvářeny tzv. doménové indexy.

Jednou z oblastí pracující s takovými komplexními daty je oblast databází pohybujících se objektů, které zpracovávají objekty měnící v čase svou polohu, případně i tvar. V minulosti byly vyvinuty mnohé metody pro přístup k časoprostorovým datům. Výzkumy se v tomto směru zaměřují na indexování již zaznamenaných (historických) dat nebo na indexování dat právě zaznamenávaných, případně jejich budoucích hodnot. Jeden ze stromů indexujících historická data je TB-strom, a právě tento strom bude využit při implementaci doménového indexu.

Cílem takto vytvořeného doménového indexu by tedy mělo být urychlení a zefektivnění práce s databází zpracovávající časoprostorová data.

Práce je strukturována do následujících osmi kapitol.

Po této úvodní kapitole následuje kapitola druhá, která se zabývá obecnými principy používání indexů v databázi Oracle. Popisuje jednotlivé typy indexů a prezentuje tak různorodé možnosti indexování dat v této databázi.

Třetí kapitola uvádí čtenáře do základů rozšířeného indexování, vysvětluje důvody jeho použití a možnosti souvisejícího frameworku. Dále charakterizuje jednotlivé indexovací struktury a používané rozhraní ODCI (Oracle Data Cartridge Interface). Při podrobnějším rozboru rozšířeného indexování se zaměřuje na prvky, které jsou pro vytvoření doménového indexu nezbytné, a na konci kapitoly je také rozebrána problematika využívání systémově řízených indexů.

Čtvrtá kapitola se zabývá obecnými zákonitostmi platnými při indexování pohybujících se objektů. Zavádí pojem trajektorie a zaměřuje se na konkrétní způsoby dotazování nad pohybujícími se objekty. Kapitola dále uvádí výčet některých přístupových metod, z nichž je následně vybrán TB-strom a detailně popsán s ohledem na budoucí implementaci.

Pátá kapitola popisuje samotnou implementaci doménového indexu. Uvádí použitý jazyk, základní typ pohybujícího se bodu a způsob jeho uložení. Dále jsou zde uvedeny konkrétní typy reprezentující TB-strom a pomocí metod pro definici, údržbu a prohledávání indexu formulován implementační typ. Na závěr jsou popsány implementované operátory a jejich konkrétní funkční implementace.

Šestá kapitola popisuje reálné použití doménového indexu, jeho vlastnosti a testování s využitím implementovaných operátorů.

V sedmé kapitole jsou prezentovány možné návrhy na vylepšení a závěrečná kapitola shrnuje výsledky celé práce.

2 Indexy v Oracle Database 11g

Databáze Oracle, nebo přesněji Oracle RDBMS (relational database management system) je SŘBD (systém řízení báze dat) vyvíjený a spravovaný společností Oracle (Oracle Corporation). Jedná se o multiplatformní databázový systém s velmi pokročilými možnostmi zpracování dat, vysokým výkonem a škálovatelností.

Nejnovější verze Oracle Database 11g podporuje nejen standardní dotazovací jazyk SQL podle normy SQL92, ale také imperativní programovací jazyk PL/SQL, proprietární firemní rozšíření Oracle, který rozšiřuje možnosti základního SQL. Oracle 11g dále podporuje objektové databáze a databáze uložené v hierarchickém modelu dat (XML databáze a jazyk XSQL). Z pohledu rozšířeného indexování může být také důležitá podpora jazyků Java a C/C++.

2.1 Základní princip indexů

Index je datová struktura, která urychluje přístup k jednotlivým řádkům databáze. Je spojený s určitou tabulkou a obsahuje data z jednoho nebo více sloupců dané tabulky. Index je důležitý v případě, kdy se vybírá pouze malá podmnožina řádků z tabulky. Je třeba upozornit, že index souvisí s fyzickou strukturou databáze, naproti tomu klíč (primární, cizí apod.) je termín pro logickou entitu, typicky hodnotu uloženou v indexu.

Základní syntaxe pro vytváření indexu v jazyce SQL je následující:

```
CREATE INDEX jméno ON tabulka (sloupec1, sloupec2, ...),
```

Kód 2.1: Vytvoření implicitního indexu nad tabulkou

kde `jméno` je název indexu, `tabulka` je název tabulky a `sloupec1`, `sloupec2` atd. jsou názvy sloupců dané tabulky.

Pokud jsou hodnoty v souvisejících sloupcích změněny, server Oracle automaticky modifikuje hodnoty v indexu. Index obsahuje méně dat než celý sloupec v tabulce a je uložen ve speciální struktuře, která umožňuje rychlejší čtení a získání dat, proto zabere menší počet vstupně - výstupních operací. Výběr řádků založený na hodnotě indexu může být tedy rychlejší než výběr řádků podle hodnot v tabulce. Druhý důvod rychlejšího přístupu je ten, že indexy jsou uloženy seřazené v určitém pořadí, ať už vzestupně, nebo sestupně (to lze definovat při vytváření indexu).

Kromě hodnot sloupce je v indexu uložen i identifikátor řádku **ROWID**, který obsahuje indexovanou hodnotu a umožňuje tak optimální přístup k určitému řádku. ROWID mohou být:

- **Fyzické** – ukládají adresy řádků do obyčejných tabulek (kromě indexově orientovaných tabulek), seskupených tabulek, oddílů a pododdílů, indexů, oddílů a pododdílů indexů.

- **Logické** – ukládají adresy řádků do indexově orientovaných tabulek.

Datový typ **UROWID** (universal ROWID) podporuje oba typy ROWID, ale i ROWID cizích tabulek mimo Oracle. Sloupec typu UROWID může ukládat všechny typy ROWID. Pokud není dostupný fyzický ROWID, obsahuje logický ROWID.

Indexy můžeme rozdělit na **jedinečné (unique)** a **duplicitní (non-unique)**.

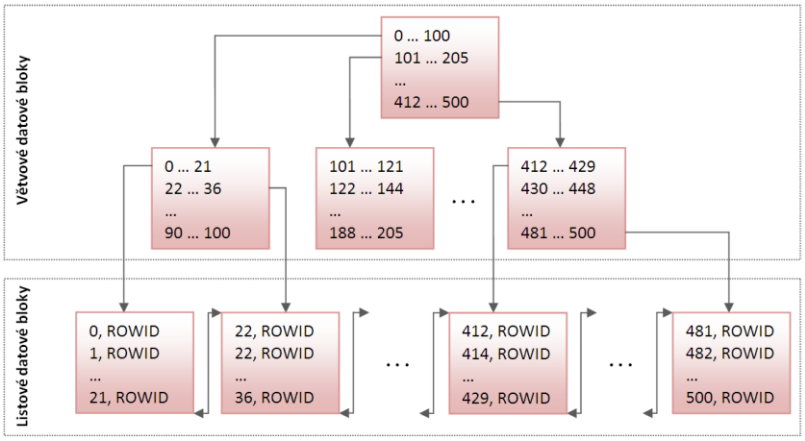
- **Jedinečný index** – žádné dva řádky v tabulce nebo pohledu nesmí mít stejnou hodnotu ve sloupcích podílejících se na indexu. V indexovaných sloupcích tabulky se tedy nebudou vyskytovat duplikované hodnoty, přičemž by měl sloupec umožnit vložit i hodnotu NULL. Tento řádek potom není zahrnut do indexu. Obvykle se využívá pro implementaci primárního klíče nad tabulkou.
- **Duplicitní index** – se vytváří bez použití klíčového slova pouze příkazem `CREATE INDEX`. Tyto indexy nevyžadují jedinečné hodnoty ve sloupci.

Oracle podporuje několik druhů indexů. Za základní lze považovat standardní B-strom, dále bitmapové indexy, indexy založené na funkci a aplikační doménové indexy. Všechny tyto způsoby indexování byly uvedeny již ve verzi Oracle 8i. Oracle 11g přináší navíc možnost používat tzv. neviditelné indexy.

2.2 B-strom

Index využívající B-strom (balanced tree) je standardním typem databázového indexu v Oracle (Přesněji se jedná o *B*strom*, který se od základního B-stromu liší tím, že data ukládá pouze v listech a tím, že minimální počet následníků není ½ N, ale ⅔ N, kde N je řád stromu [15]. V oficiální dokumentaci je však nazýván obecně - *B-strom*, proto se dál budu držet tohoto názvu). Index v podobě B-stromu je vlastně seřazený seznam hodnot rozdělených do určitých intervalů.

B-strom v Oracle se skládá z jedné nebo více úrovní větvových bloků sloužících pro vyhledávání a jedné (nejnižší) úrovně listových bloků pro data. Rodičovské bloky obsahují ukazatele na jejich následníky, které obsahují hodnoty klíčů, přičemž počet klíčů a ukazatelů je omezen velikostí bloku. Počet úrovní větví mezi kořenem a listy bývá nazýván jako hloubka indexu. Listy obsahují každou indexovanou hodnotu a odpovídající ROWID pro konkrétní řádek tabulky a jsou vzájemně propojeny do obousměrného lineárního seznamu (viz *Obr. 2.1*).



Obr. 2.1: Index B-stromu

Na obrázku má nejnižší větev první záznam *0 ... 100*, který ukazuje na nejlevějšího potomka. Potomek pak obsahuje záznamy *0 ... 21*, *22 ... 36*, ..., z nichž každý ukazuje opět níže, tentokrát už na listy. Ty obsahují hodnoty všech klíčů spadajících do daného intervalu.

Ve vyšších úrovních větvových bloků neobsahuje struktura B-stromu příliš mnoho bloků, proto přístup i do relativně větší hloubky stromu nezabere příliš mnoho V/V operací. Listové bloky mají všechny stejnou hloubku v indexu, proto jejich získání zabere vždy stejný počet V/V operací. Počet V/V operací pro získání listového bloku předurčuje výkonnost indexu.

Oracle povoluje vytvářet indexově orientované tabulky, ve kterých listové bloky, spíše než ROWID, které ukazuje na určitý řádek, ukládají celý řádek. Tím indexově orientované tabulky redukují množství prostoru potřebného k uložení dat. Na druhou stranu nemůže být v těchto tabulkách použito omezení UNIQUE a nemohou být uloženy v clusteru. Navíc nepodporují distribuci, replikaci a rozdělování dat, ačkoliv od verze Oracle 10g je možné indexově orientované tabulky použít pro zachycování a aplikaci změn v datových tocích v Oracle (Oracle streams).

2.3 Indexy s reverzním klíčem

Indexy s reverzním klíčem (reverse key indexes), jak již jejich název napovídá, automaticky obrací pořadí bajtů v hodnotách klíčů indexu. Tento typ indexů se využívá především v prostředí OLAP (online analytical processing).

Reverzní klíč je možné vytvořit pomocí klíčového slova `REVERSE` příkazu `CREATE INDEX`.

Pro lepší pochopení potřeby použití indexů s reverzním klíčem je vhodné si shrnout základní fakta o B-stromech. Především je důležité to, že hloubka B-stromu je určena počtem záznamů v listových blocích. Čím větší je hloubka stromu, tím více je ve stromu větvových bloků a tím více V/V operací je potřeba pro přístup k danému listovému bloku.

Problémem je, že při vytváření B-stromu existují hodnoty, které porušují jeho vyváženost. Mezi takové hodnoty patří inkrementální hodnoty, jako jsou vzestupná posloupnost čísel nebo po sobě jdoucí data. Tyto hodnoty jsou vždy přidávány na pravou stranu indexu, kde se tím pádem

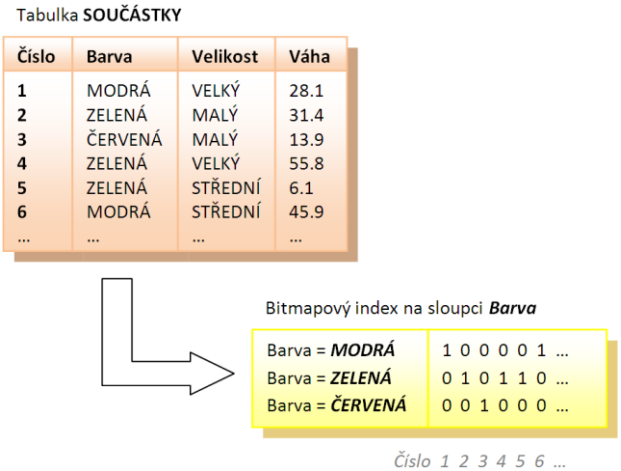
hodnoty neustále zvyšují. Navíc, při mazání ze starších dat z indexu, začíná být strom na levé straně nevyvážený. To má za následek, že se vyvážený B-strom stane po nějakém čase stromem nevyváženým, ve kterém se listy se starými daty na levé straně vyskytují řidčeji než na straně pravé, kde se nacházejí nová data. Tento nevyvážený růst vede k zbytečně hlubokému B-stromu a tomu je třeba zabránit.

Problém se dá vyřešit buď pravidelným mazáním a opětovným vytvářením indexu nebo použitím indexu s reverzním klíčem. Obrácení klíče přinese rovnoměrnější rozložení po všech listových blocích, což snižuje soupeření o prostředky v případě, kdy provádí vkládání nových řádků více uživatelů současně. Tento typ indexu také snižuje pravděpodobnost vzniku úzkých míst v OLTP prostředí tehdy, kdy jsou data vyhledávána nebo aktualizována již krátce po vložení.

2.4 Bitmapové indexy

Na rozdíl od standardního B-stromu, kde jsou ROWID uloženy v listových blocích, v bitmapovém indexu (bitmap index) každý bit indexu reprezentuje ROWID. Index ukládá řetězec takových bitů pro každou možnou hodnotu indexovaného sloupce. Pokud se daná hodnota v daném sloupci vyskytuje, pak je daný bit v indexu nastaven na 1. Délka tohoto řetězce v indexu je stejná jako počet řádků v indexované tabulce.

Pokud je počet různých hodnot nízký (viz *Obr. 2.2*), je pro uložení bitmapového indexu potřeba mnohem méně místa než pro uložení B-stromu. Kromě významné úspory místa může použití bitmapového indexu výrazně urychlit dobu odezvy, protože Oracle může velmi rychle odstranit potenciální řádky z dotazu, který obsahuje vícenásobné klauzule `WHERE`, ještě před vlastním přístupem do tabulky.



Obr. 2.2: Bitmapový index v Oracle

O použití bitmapového indexu je vhodné uvažovat v případě, kdy počet různých hodnot ve sloupci je méně než 1 % počtu řádků v tabulce nebo pokud se hodnoty ve sloupcích opakují více než 100 krát.

Bitmapové indexy jsou důležité především v datových skladech, ve kterých každá dimenze obsahuje mnoho opakujících se hodnot, a fronty typicky vyžadují interakci více dimenzí.

Bitmapové spojované indexy

Tento typ bitmapových indexů vytváří bitmapový index nad sloupcem tabulky, který bývá často spojován pomocí operace `JOIN` s jednou nebo více tabulkami nad stejnými sloupci. Z toho plynou výrazné výhody v prostředí datových skladů, kdy jsou bitmapové spojované indexy vytvářeny nad základní tabulkou a jednou nebo více rozměrovými tabulkami. Při vytvoření indexu se zároveň vytvoří struktura spojení tabulek, což ušetří jak prostředky procesoru, tak i V/V operace při aktuálním spojování tabulek.

2.5 Indexy založené na funkci

Indexy založené na funkci (function-based indexes) se podobají standardnímu B-stromu nebo bitmapovému indexu, s tím rozdílem, že transformace sloupce nebo sloupců je uložena v indexu, a ne v samotných sloupcích. Index založený na funkci vypočítá hodnotu funkce či výrazu a uloží ji do indexu.

Funkce používaná pro výpočet hodnoty indexu může být aritmetický výraz nebo výraz obsahující PL/SQL funkci, SQL funkci nebo například externí C funkci. Výraz nesmí obsahovat agregační funkci a musí být deterministický. Takže místo indexování např. sloupce „*jméno*“ indexujeme sloupec založený na funkci jako například „UPPER(*jméno*)“

Indexy založené na funkci jsou užitečné třeba v případech, kdy jsou jména nebo adresy uloženy v databázi tak, že používají různou velikost písmen. Normální index nad sloupcem, který obsahuje například hodnotu „Karel“ nevrátí žádnou hodnotu v případě, že se vyhledává jméno „karel“. Pokud byl ovšem index nad sloupcem *jméno* vytvořen tak, že se indexuje jméno se všemi písmeny velkými, musejí být všechny vyhledávací dotazy příslušně upraveny pro vyhledávání jména jen s velkými písmeny.

Tato schopnost nabývá ještě většího významu v situacích, kdy si v Oracle database vytváříme vlastní, mnohdy velmi komplikované funkce, které mohou při použití s indexy založenými na funkci značně ovlivnit výkonnost celé databáze.

2.6 Doménové indexy

Doménový index je speciální typ indexu vytvořený pro určitou doménu, jako jsou například dokumenty, prostorová data, obrázky či videoklipy. Jedná se o uživatelskou datovou strukturu, která umožňuje efektivní přístup k těmto datům. Doménovým indexům je možné přiřadit statistiky a cenové funkce. Optimalizátor, který určuje nejvýhodnější cestu vedoucí k výsledku dotazu, pak tyto

indexy dokáže používat v optimalizacích stejným způsobem jako vestavěné indexy (podrobněji jsou doménové indexy rozebrány dále).

2.7 Neviditelné indexy

Pro všechny výše popsané typy indexů byla ve verzi Oracle 11g představena možnost určení tzv. neviditelných indexů (invisible indexes). Neviditelný index se používá v případech, kdy chceme, aby byl index ignorován optimalizátorem, aniž bychom jej museli mazat nebo přepínat do offline režimu. Pokud je index neviditelný, není uvažován jako možný krok v přístupové cestě, ale aktualizace a mazání obsažených dat může i nadále probíhat standardním způsobem.

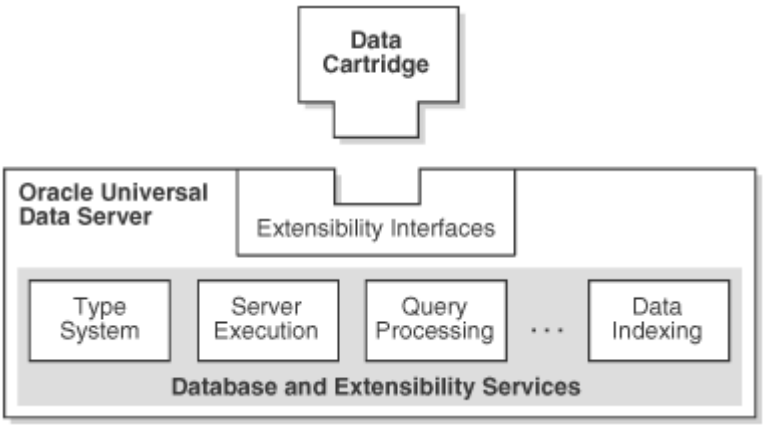
Funkce neviditelných indexů se používá například při testování, kde mnohdy potřebujeme index pouze dočasně vyjmout ze zpracovávání.

3 Rozšířené indexování v Oracle Database 11g

Rozšířené indexování (extensible indexing) umožňuje implementovat a následně použít i jiné způsoby indexování než jen ty nativně podporované v Oracle. Rozšířené indexování se v Oracle používá v rámci speciálních komponent, tzv. ***datových cartridgí*** (data cartridges), což jsou softwarové balíčky, které rozšiřují možnosti Oracle serveru a předurčují, jak bude server interpretovat, ukládat, získávat nebo indexovat aplikační data.

Základní vlastnosti datových cartridgí jsou:

- Jsou *server-based* (založené na serveru) – jejich prvky jsou uložené na serveru nebo jsou ze serveru přístupné. Server také spouští všechny procesy v cartridgi, případně je posílá jako externí procedury.
- Rozšiřují server – definují nové typy a chování, které jsou jinak serveru nedostupné a používají tyto nové funkce ve svých aplikacích.
- Jsou integrovány se serverem – pomocí rozhraní.
- Jsou v balíčku – k databázi jsou připojovány jako celek.



Obr. 3.1: Služby Oracle (převzato z [16])

Na *Obr. 3.1* je znázorněno schéma propojování datové cartridge s Oracle serverem pomocí rozhraní pro rozšiřování (*Extensibility Interface*). Oracle server obsahuje databázové a rozšiřující služby zahrnující například indexování dat, zpracování dotazů a další.

Z pohledu návrhových vzorů se v případě datové cartridge jedná o návrhový vzor *Factory (Továrna)*, která vytváří určitý typ indexu v podobě vzoru *Facade (Fasáda)*. Tento index je pak konkrétněji nazýván jako doménový index.

Databáze musí neustále zpracovávat nové, komplexnější typy dat (multimediální, zdravotnická apod.), specifické pro určité úlohy. S tím roste i složitost dotazů a objem dat, který je třeba prohledat. Proto Oracle poskytuje framework pro rozšířené indexování, který umožňuje upravovat indexovací metody podle potřeby jednotlivých aplikací a zlepšit tak výkon a zajistit snazší používání.

Při rozšířeném indexování sama aplikace definuje struktury indexu, ukládá indexová data, ať už dovnitř, nebo vně Oracle databáze, sama je spravuje a získává při vyhodnocování dotazů. Tím tato aplikace kontroluje strukturu a sémantiku obsahu indexu.

Databázový systém spolupracuje s aplikací na vytváření, udržování a používání doménového indexu. Proto je možné vytvářet indexy pro provádění úkolů specifických pro určitou doménu a skládat normální dotazy s nově definovanými operátory.

3.1 Kdy použít rozšířené indexování

Metody indexování vestavěné v Oracle sice pokrývají velké množství situací, ale se stále komplexnějšími daty a s doménově specifickými aplikacemi roste potřeba i jiných přístupů, proto existuje rozšířené indexování, které může pomoci řešit například takovéto problémy:

- Implementace nových operátorů pro vyhledávání pomocí specializovaných indexových struktur. – Můžeme definovat operátory pro specializované vyhledávání pomocí vlastních indexových struktur.
- Indexování nestrukturovaných dat. – Vestavěné prostředky neumí indexovat sloupec obsahující LOB hodnoty.
- Indexování atributů sloupcových objektů. – Pomocí vestavěných prostředků také nelze indexovat sloupcové objekty ani části kolekcí.
- Indexování hodnot získaných z doménově specifických operací. – Objektové typy v Oracle mohou být porovnávány pomocí mapovacích nebo řadicích funkcí. Pokud objekt používá mapovací funkci, můžeme sice pro vyhodnocování relačních predikátů definovat index založený na funkci. Tento přístup však funguje pouze v případě, kdy mají predikáty konečný rozsah parametrů, jelikož musí být možné spočítat funkční hodnoty všech řádků. Navíc řadící funkce nemůžeme použít pro vytváření indexu. Proto je rozšířené indexování vhodné i v tomto případě.

3.2 Indexové struktury pro rozšířené indexování

Různá data mohou být indexována různými způsoby. Aby byl přístup k datům co nejefektivnější, je potřeba zvolit správný typ indexování, protože ne každý typ indexování je vhodný pro danou aplikaci. Databázové aplikace získávají podmnožiny dat pomocí dotazování nad indexy. Takové dotazy se

mohou často lišit v operátoru, který je vyjadřuje a tak i nejvhodnější metody indexování mohou být různé.

Například pro zjištění, kteří zaměstnanci pracují v Brně, je třeba operátor testující shodu. V takovém případě je vhodné použití hashovací struktury, která s těmito operátory pracuje velmi efektivně. Naopak pro zjištění zaměstnanců, jejichž plat je v určitém rozmezí, je třeba operátor pro testování rozsahů a k tomu je lepší použití struktury B-stromu.

Podrobný popis indexovacích algoritmů lze najít například v [15].

3.3 Framework pro rozšířené indexování

Framework pro rozšířené indexování je rozhraní založené na jazyce SQL, které umožňuje definici doménově specifických operátorů a indexovacích schémat a jejich integraci do serveru Oracle. Framework se skládá z následujících komponent:

- **Indexové typy** – indexový typ je objekt, ve kterém jsou pomocí rozhraní *ODCIIndex* specifikovány postupy, které se starají o definici, údržbu a vyhledávací operace pro aplikačně specifické indexy. Indexový typ říká Oracle serveru jak založit uživatelem definovaný index na určitém sloupci tabulky nebo atributu objektu.
- **Doménové indexy** – index specifický pro určitou aplikaci vytvářený za pomoci indexového typu. Doménový se nazývá proto, že indexuje data v aplikačně specifické doméně. Doménový index je instance indexu, který je vytvářen, řízen a přistupován pomocí postupů specifikovaných v indexovém typu.
- **Operátory** – příkazy pro dotazování a manipulaci s daty mohou používat aplikačně specifické operátory, jako je například operátor `Overlaps` v prostorové doméně. Uživatelsky definované operátory jsou svázány s funkcemi. Mohou být také vyhodnocovány pomocí indexů. Například operátor rovnosti může být vyhodnocen pomocí hashovaného indexu.
- **Indexově orientované tabulky** – pomocí indexově orientovaných tabulek může aplikace definovat, vytvářet, spravovat a přistupovat indexy komplexních objektů. Z pohledu aplikace je index implementován jako tabulka, kde každý řádek je záznam v indexu. Indexově orientovaná tabulka zvládne i duplikované záznamy v indexu, což může být u komplexních typů dat důležité.

Pomocí frameworku pro rozšířené indexování můžeme vytvořit doménový index, který funguje podobně jako kterýkoliv jiný index v Oracle. Uživatelé mohou používat standardní dotazy s využitím programátorem definovaných operátorů. Pro vytváření, mazání, vyprazdňování, změnu a vyhledávání doménového indexu použije server Oracle patřičnou část kódu aplikace, který programátor specifikoval jako součást indexového typu.

Framework pro rozšířené indexování umožňuje především:

- Zapouzdření aplikačně specifických postupů pro správu indexu do indexového typu.
- Definici doménového indexu na sloupcích tabulky.
- Efektivní zpracování aplikačně specifických operátorů.

3.4 Rozhraní ODCIIndex

Rozhraní ODCIIndex (Oracle Data Cartridge Interface Index) specifikuje všechny postupy, které musíme dodat pro vytvoření indexového typu. Zahrnuje následující skupiny metod.

- Metody pro definici indexu
- Metody pro správu indexu
- Metody pro prohledávání indexu
- Metody pro metadata indexu

3.4.1 Metody pro definici indexu

Metody pro definici indexu jsou volány v případě použití příkazů `CREATE`, `ALTER`, `DROP` nebo `TRUNCATE` na index daného indexového typu.

- **`ODCIIndexCreate()- p`**okud uživatel zadá příkaz `CREATE INDEX`, který odkazuje na daný indexový typ, Oracle zavolá metodu `ODCIIndexCreate()`, předá ji parametry specifikované uživatelem a popis indexu. Tato metoda se obvykle používá pro vytváření tabulek nebo souborů, ve kterých je plánováno ukládání indexových dat.
- **`ODCIIndexAlter()`** – pokud uživatel zadá příkaz `ALTER INDEX` odkazující na daný indexový typ, Oracle zavolá metodu `ODCIIndexAlter()`, předá jí popis doménového indexu, který má být změněn spolu s parametry. Další detaily závisí na charakteru doménového indexu.
- **`ODCIIndexDrop()`** – používá se ke mazání indexu daného indexového typu a je volána po příkazu `DROP INDEX`.

3.4.2 Metody pro správu indexu

Metody pro správu indexu jsou volány v případě použití příkazů `INSERT`, `UPDATE` a `DELETE` na tabulky indexované daným indexovým typem.

- **`ODCIIndexInsert()`** – pokud uživatel vloží do tabulky záznam, případně více záznamů, Oracle zavolá metodu `ODCIIndexInsert()` s novými hodnotami indexovaných sloupců ve vkládaných řádcích a odpovídajícím ROWID.

- **`ODCIIndexDelete()`** – pokud uživatel smaže záznam, Oracle tuto metodu zavolá se starými hodnotami indexovaných sloupců mazaného řádku a odpovídajícím ROWID.
- **`ODCIIndexUpdate()`** – pokud uživatel aktualizuje záznam, Oracle tuto metodu zavolá zároveň se starými i novými hodnotami indexovaných sloupců aktualizovaného řádku včetně odpovídajícího ROWID.

3.4.3 Metody pro prohledávání indexu

Pomocí těchto metod se specifikuje indexová implementace pro vyhodnocování predikátů obsahujících operátory podporované daným indexovým typem. Jedná se o metody pro inicializaci, získávání řádků nebo ROWID a úklidu poté, co jsou požadované řádky získány.

Způsoby pro vyhodnocení operátorových predikátů a následné vrácení výsledného souboru řádků jsou dva:

- **Předpočítání všeho** – vypočítá kompletní sadu výsledků v `ODCIIndexStart()`. Opakuje se na výsledku a z každého volání `ODCIIndexFetch()` vrací skupinu řádků. Tento způsob je vhodný pro operátory, které se musí zaměřit na celou sadu výsledků, aby mohly vypočítat hodnocení, relevanci atp. pro každý řádek. Pokud to aplikace vyžaduje, je také možné vrátit na požádání pouze jeden řádek.
- **Inkrementální výpočet** – v každém volání `ODCIIndexFetch()` vypočítá skupinu výsledných řádků. Tento způsob výpočtu je vhodný pro operátory, které mohou určit řádky jeden po druhém, aniž by musely pracovat s celou sadou výsledků. I tímto postupem je možné vrátit pouze jeden řádek.

Jednotlivé metody jsou:

- **`OCDIIndexStart()`** – Oracle tuto metodu zavolá na začátku prohledávání indexu a předá jí informace o indexu a operátoru. Metoda nejdříve inicializuje struktury používané k prohledávání. Potom parsuje a vykoná SQL příkaz, který se dotazuje nad tabulkou s indexovými daty. Dále uloží informace požadované metodami pro získávání dat nebo čištění a nakonec vrátí stav vyhodnocování operátoru nebo handle na něj.

  Informace o indexu a operátoru nejsou těmto metodám posílány, proto musí `ODCIIndexStart()` uložit stavová data, která potřebují být sdílená vyhledávacími metodami, a vrátit je pomocí výstupního parametru `stxc`.

  Metody `ODCIIndexFetch()` a `ODCIIndexClose()` mohou ke stavovým datům přistupovat pomocí parametru `SELF`.
- **`ODCIIndexFetch()`** – Oracle zavolá tuto metodu pro vrácení více ROWID skupiny řádků, které vyhovují operátorovému predikátu, a předá jí stavová data vrácená

voláním metod `ODCIIndexStart()` nebo `ODCIIndexFetch()`. Operátorový predikát je specifikován jménem, atributy a také horní a dolní hranicí vracených hodnot. Proto musí `ODCIIndexFetch()` vrátit ROWID řádků jejichž hodnota spadá do vymezených hranic. Na konci prohledávání je vrácena hodnota `NULL`.

- **`ODCIIndexClose()`** – tato metoda je volána pro ukončení zpracovávání operátoru a měla by vykonat potřebné uzavírací a úklidové operace, které indexový typ vyžaduje.

3.4.4 Metody pro metadata indexu

Zde se jedná pouze o jednu volitelnou metodu **`ODCIIndexGetMetadata()`**. Pokud je metoda implementována, používá se pro zápis implementačních metadat do souboru výpisu (export dump file). Metadaty mohou být informace o metodách, verzi, individuálním nastavení atd.

Podrobnější popis všech metod včetně jejich syntaxe, popisu parametrů, návratových hodnot a poznámek k použití lze nalézt v [16].

3.4.5 Transakční sémantika při vykonávání indexových metod

Všechny popsané metody (kromě metod pro definici indexu) jsou volané stejnou transakcí, která tyto akce spouští. Takže všechny akce prováděné těmito metodami jsou atomické a jsou prováděny nebo rušeny na základě rodičovských transakcí. Proto platí následující pravidla:

- Metody pro definici indexu nemají žádná omezení.
- Metody pro správu indexu mohou vykonávat pouze DML příkazy. Tyto příkazy nemohou aktualizovat základní tabulku, na které je vytvořen doménový index.
- Metody pro prohledávání indexu mohou vykonávat pouze SQL příkazy.

Metody pro definici indexů nemají žádná omezení vnitřních akcí. Typický postup v `ODCIIndexCreate()` může být například takový, že nejdříve vytvoříme indexovou tabulku, pak se do ní vloží data a nakonec se vytvoří sekundární index nad jedním nebo více sloupci této indexové tabulky.

Pro to, aby metoda `ODCIIndexCreate()` vykonala sekvenci DDL a DML příkazů, musí být každý příkaz uvažován jako nezávislá operace. Proto není garantovaná atomičnost změn prováděných pomocí `ODCIIndexCreate()`. To samé platí pro všechny další metody sloužící k definici indexů.

3.5 Indexový typ

Indexový typ (*INDEXTYPE*) vytváří nové indexovací schéma tím, že specifikuje sadu podporovaných operátorů a metod, které spravují doménový index. Indexový typ zapouzdřuje metody vyhledávání a získávání dat pro celou konkrétní doménu. Indexový typ je podobný indexům nativně poskytovaným v databázi Oracle. Rozdíl je v tom, že je třeba zajistit aplikační software, který indexový typ implementuje.

Indexový typ se skládá ze dvou hlavních částí:

- **Metody**, které implementují chování indexového typu, například vytváření a prohledávání indexu.
- **Operátory** podporované indexovým typem, jako např. `Contains()` nebo `Overlaps()`.

3.5.1 Postup při vytváření indexového typu

Při vytváření indexového typu je zapotřebí provedení určitých kroků a vytvoření jednotlivých komponent v určitém pořadí. To závisí na tom, zda vytváříme funkční implementaci založenou na indexu, či nikoliv.

Postup pro funkční implementaci nezaloženou na indexech:

1. Definovat a naprogramovat funkce pro podporované operátory.
2. Vytvořit nový operátor a svázat jej s implementací funkce.
3. Vytvořit metody implementující rozhraní ODCIIndex a definovat typ, který je zapouzdřuje, tzv. **implementační typ**.
4. Vytvořit indexový typ specifikováním implementačního typu a vypsáním operátorů s jejich vazbami.

Postup pro funkční implementaci založenou na indexech:

Postup je podobný, jako předchozí, jen jednotlivé kroky jsou prováděny v jiném pořadí, protože funkční implementace založená na indexech vyžaduje implementační typ jako parametr.

1. Definovat implementační typ.
2. Definovat a naprogramovat funkce.
3. Vytvořit operátor.
4. Vytvořit indexový typ.

3.5.2 Operace s indexovými typy

Mezi operace s indexovými typy lze zařadit způsob vytváření, mazání a komentování indexových typů.

Vytváření indexových typů

Pokud je rozhraní ODCIIndex úspěšně naimplementováno a je definovaný implementační typ, můžeme vytvořit nový indexový typ tím, že specifikujeme seznam podporovaných operátorů a odkážeme na typ, který implementuje rozhraní daného indexu.

Následuje příklad, kde je pomocí DDL příkazů definován nový indexový typ `IType`, který podporuje operátory `Vetsi` a `Mensi` a jehož implementace je poskytována typem `DomIndex`. Nakonec je specifikována Oraclem doporučovaná kombinace systémově řízených úložných tabulek a lokálních doménových indexů (viz dále).

```
CREATE INDEXTYPE IType
FOR
  Vetsi (VARCHAR2, VARCHAR2),
  Mensi (VARCHAR2, VARCHAR2)
USING DomIndex
WITH LOCAL PARTITION
WITH SYSTEM MANAGED STORAGE TABLES;
```

Kód 3.1: Vytvoření indexového typu

Do rozhraní ODCIIndex musí implementační typ navíc implementovat metodu `ODCIGetInterfaces()`. Ta vrací verzi rozhraní specifikovaného implementačním typem a Oracle ji spouští po vykonání příkazu `CREATE INDEXTYPE`. V případě použití Oracle verze 9i a vyšší, musí v parametru `OUT` vracet hodnotu `SYS.ODCIINDEX2`.

Mazání indexových typů

Pro mazání indexových typů se využívá příkaz `DROP`. Pro předchozí příklad by mazání vypadalo takto:

```
DROP INDEXTYPE IType;
```

Kód 3.2: Smazání indexového typu

Standardní chování příkazu `DROP` je `DROP RESTRICT`, což znamená, že pokud existuje jeden nebo více doménových indexů používající daný indexový typ, pak je operace `DROP` zamítnuta. Uživatelé mohou toto přednastavené chování přepsat volbou `FORCE`, která smaže celý indexový typ a zruší veškeré jeho vazby.

Komentování indexových typů

Komentování se používá k dodání informací o indexovém typu nebo operátoru a provádí se příkazem `COMMENT`.

V souvislosti s příkladem bychom mohli komentovat takto:

```
COMMENT ON INDEXTYPE
IType IS 'Toto je pouze ukázkový příklad indexypu.';
```

Kód 3.3: Komentování indexového typu

Komentáře indexových typů lze prohlížet v následujících pohledech:

- `ALL_INDEXTYPE_COMMENTS` – komentáře pro uživatelské indexové typy přístupné danému uživateli
- `DBA_INDEXTYPE_COMMENTS` – komentáře pro všechny uživatelské indexové typy v databázi
- `USER_INDEXTYPE_COMMENTS` – komentáře pro uživatelské indexové typy náležící danému uživateli

Jednotlivé pohledy mají sloupce `OWNER` (majitel indexového typu), `INDEX_TYPE` (jméno indexového typu) a `COMMENTS` (samotný komentář).

Aby bylo možné komentovat, musí být indexový typ v našem vlastním schématu nebo musí mít nastavené právo `COMMENT ANY INDEXTYPE`.

3.6 Operátory

Uživatelem definované operátory jsou objekty na nejvyšší úrovni. V mnoha směrech fungují stejně jako vestavěné operátory (<, >, =). Mohou být volané ve stejných situacích. Přispívají ke zjednodušení SQL příkazů.

Operátory jsou identifikované jmény, která jsou ve stejných jmenných prostorech jako tabulky, pohledy, typy a funkce. Jsou svázané s funkcemi, které definují chování daného operátoru a také jsou často asociované s konkrétními indexovými typy.

3.6.1 Vazby operátorů

Vazby operátorů přiřazují operátoru signaturu funkce, která daný operátor implementuje. Signatura sestává ze seznamu datových typů argumentů funkce a návratového typu funkce. Vazby operátorů sdělují, která funkce má být při vyvolání určitého operátoru spuštěna. Operátor může být svázán s více funkcemi, pokud mají různou signaturu. Funkce proto musí mít odlišný seznam argumentů. Pokud tomu tak není, pak takové funkce nemohou být svázány s jedním operátorem, třebaže jsou jejich návratové typy rozdílné.

Operátory mohou být svázány se samostatnými funkcemi, funkcemi z různých balíčků nebo uživatelsky definovanými členskými metodami, a to v jakémkoliv dostupném schématu, přičemž každý z nich musí mít alespoň jednu vazbu.

3.6.2 Práce s operátory

Vytváření operátorů

Operátory se vytváří pomocí příkazu `CREATE OPERATOR`. V následujícím příkladu je vytvořen operátor `r_query()` a svázán s funkcí `pt_insidet()`.

```
CREATE OPERATOR r_query
BINDING (pointsT, pointsS) RETURN trajectory_entries_nt
USING pt_inside;
```

Kód 3.4: Vytvoření operátoru

Mazání operátorů

Příkazem `DROP OPERATOR` smažeme zadaný operátor a všechny jeho vazby, například:

```
DROP OPERATOR r_query;
```

Kód 3.5: Smazání operátoru

Mazání je defaultně `RESTRICT`, takže pokud je operátor určitým způsobem vázaný, je mazání zamítnuto. Toto chování můžeme změnit použitím `FORCE` (`DROP OPERATOR Vetsi FORCE`).

Změna operátorů

Použitím `ALTER OPERATOR` můžeme přidávat nebo rušit vazby existujících operátorů. V následujícím příkladu je přidána vazba operátoru `r_query()` na funkci `pt_inside_plus()`.

```
ALTER OPERATOR r_query
ADD BINDING (NUMBER, NUMBER) RETURN NUMBER
USING pt_inside_plus;
```

Kód 3.6: Změna operátoru

Pro změnu operátoru platí určitá omezení. Můžeme měnit existující operátory, přidávat nebo mazat můžeme pouze jednu vazbu, dále nesmíme mazat vazbu operátoru, pokud je jeho jediná. Pokud přidáme vazbu operátoru, který je přiřazený v určitém indexovém typu, pak tato vazba není asociována s indexovým typem, dokud nezavoláme `ALTER INDEXTYPE ADD OPERATOR`.

Komentování operátorů

Komentáře operátorů se specifikují v příkazu `COMMENT`. Například takto:

```
COMMENT ON OPERATOR
pt_inside IS 'Zjistí, zda je bod uvnitř.';
```

Kód 3.7: Komentování operátoru

Komentáře jsou podobně jako indexové typy dostupné v pohledech `USER_OPERATOR_COMMENTS`, `ALL_OPERATOR_COMMENTS` a `DBA_OPERATOR_COMMENTS`.

3.6.3 Volání a vyhodnocování operátorů

Stejně jako u vestavěných operátorů jsou uživatelsky definované operátory volané všude tam, kde se vyskytne příslušný výraz. To může být například při výběru příkazem `SELECT`, v podmínce klauzule `WHERE` nebo v `ORDER BY` a `GROUP BY`. V případě zavolání operátoru jej Oracle vyhodnotí spuštěním funkce, která je s ním svázaná. Pro případ, kdy má operátor takovýchto vazeb více, se používá přetěžování operátorů a Oracle vykoná tu, jejíž datový typ a počet argumentů je shodný s volanými.

Operátory vyskytující se mimo klauzuli `WHERE` v podstatě zastupují funkci, která je implementuje. To znamená, že význam takového operátoru je dán jeho funkční implementací. Naopak operátory použité spolu s klauzulí `WHERE` mohou být vyhodnocovány jak pomocí jejich funkční implementace, tak i metodami pro prohledávání indexů.

Při vyhodnocování operátoru v klauzuli `WHERE`, rozhoduje o použití doménového indexu vestavěný optimalizátor, který zajistí použití doménového indexu pouze v případě, pokud má sloupec využívaný operátorem nad sebou definovaný příslušný index, zároveň je tento index příslušného indexového typu a podporuje-li tento indexový typ vyhodnocovaný operátor. Pokud některá z těchto podmínek není splněna, pak musí být dotazovaná tabulka prohledána celá (tzv. full-table scan). Vyhodnocování operátoru pomocí doménového indexu probíhá s využitím výše popsaných metod pro prohledávání indexu, a to tak, že se nejdříve zavolá metoda `ODCIIndexStart()` pro každý jeden použitý operátor spolu s informacemi souvisejícími s daným operátorem. Poté se pomocí metody `ODCIIndexFetch()` získají ROWID požadovaných řádků a nakonec metoda `ODCIIndexClose()` dokončí vyhodnocování operátoru.

Naopak operátory použité mimo klauzuli `WHERE` mohou být vyhodnocovány pouze pomocí funkční implementace. Funkční implementace také však může využít doménové indexy. V takovém případě je ale vhodné dodat stávající funkci ještě tři další argumenty, a to:

- *Index context* – informace o doménovém indexu a ROWID vyhodnocovaných řádků.
- *Scan context* – hodnota sloužící pro předávání stavu mezi jednotlivými voláními operátoru nad jednotlivými řádky.
- *Scan flag* – příznak posledního volání operátoru.

Také je třeba při svazování operátoru s jeho funkční implementací použít navíc klauzuli

„*…WITH INDEX CONTEXT, SCAN CONTEXT DomIndex…", kde DomIndex* je související implementační typ.

3.7 Doménové indexy

Databázový server Oracle při vytváření, údržbě a vyhledávání doménového indexu spolupracuje s aplikací. Samotná indexová struktura může být uložena jako indexově orientovaná tabulka nebo externě jako soubor.

Oracle Database 11g přináší nový **systémově řízený** přístup k doménovým indexům, který oproti **uživatelsky řízenému** přístupu odstraňuje programové zbytečnosti a zvyšuje výkonnost indexů (viz *kap. 3.7.7*).

Uživatel si může doménový index určitého typu vytvořit, ale až poté, co je vytvořen příslušný indexový typ.

3.7.1 Operace s doménovými indexy

Mezi tyto operace patří vytváření (create), změna (alter), mazání záznamů (truncate) a mazání (delete) indexů.

Vytváření doménových indexů

Doménový index se vytváří nad sloupcem tabulky jako například standardní B-strom, ale navíc musí být specifikován indexový typ. V návaznosti na předchozí příklady může být doménový index vytvořen například takto:

```
CREATE INDEX idx1 ON trajTable(trajectory)
INDEXTYPE IS IType
PARAMETERS ('test;');
```

Kód 3.8: Vytvoření doménového indexu

Slovo `INDEXTYPE` specifikuje určitý indexový typ. `PARAMETERS` popisuje parametry doménového indexu v podobě řetězce, který je následně předán metodě `ODCIIndexCreate()`.

Změna doménových indexů

Změna se provádí příkazem `ALTER INDEX`. Parametry jsou přeposílány metodě `ODCIIndexAlter()`, která provádí změnu indexu. V následujícím příkladu (*Kód 3.9*) je nejdříve změněn název indexu, poté změněn parametr, obsahující informace o způsobu zaokrouhlování, na novou hodnotu.

```
ALTER INDEX idx1 RENAME TO MyIndex;
ALTER INDEX idx1 REBUILD
PARAMETERS('zaokrouhlovat:desetiny;');
```

Kód 3.9: Změna doménového indexu

Mazání záznamů doménových indexů

Pro vyprázdnění doménového indexu není speciální příkaz. Pokud je odpovídající základní tabulka vyprázdněna (truncate), pak je také vyprázdněna související tabulka s doménovým indexem a je zavolána metoda `ODCIIndexAlter()`.

```
TRUNCATE TABLE trajTable
```

Kód 3.10: Smazání dat doménového indexu

Tento příkaz zavolá metodu `ODCIIndexAlter()` s parametrem `alter_option` nastaveným na `AlterIndexRebuild` a tím se vyprázdní související doménový index (v tomto případě `idx1`).

Mazání doménových indexů

K vymazání celého doménového indexu použijeme příkaz `DROP INDEX`. Například:

```
DROP INDEX idx1;
```

Kód 3.11: Smazání doménového indexu

Zavolá se metoda `ODCIIndexDrop()` spolu s informacemi o indexu.

3.7.2 Stavy doménových indexů

Doménové indexy mohou být v jednom nebo více určitých stavech, a to:

- `IN_PROGRESS` – v tomto stavu se index nebo jeho část nachází při vykonávání nějaké metody z rozhraní ODCIIndex. Tento stav bývá zpravidla dočasný, avšak pokud provádění metody skončí předčasně, index může zůstat takto označen.
- `FAILED` – pokud nějaká metoda z rozhraní ODCIIndex vykonávající DDL operaci na indexu vrátí chybu, je index nebo jeho část označena jako `FAILED`.
- `UNUSABLE` – index je takto označen v případě provádění určitých operací údržby. Pro rozdělené indexy je `UNUSABLE` spojeno s jedním oddílem, nikoliv celým indexem.
- `VALID/INVALID` – označení `VALID/INVALID` se použije, pokud je objekt, na kterém je index přímo nebo nepřímo závislý, existuje a je platný, resp. je smazán a je zrušen. `VALID/INVALID` se používá pouze pro celý index, nikdy pro jeho oddíl.

3.7.3 Doménové indexy indexově orientovaných tabulek

Pokud indexovým typem vytváříme doménový index na indexově orientované tabulce, můžeme pro stanovení, že bude základní tabulka indexově orientovaná, použít bit `IndexOnIOT` z `IndexIntoFlags` ze struktury `ODCIIndexInfo`.

Pokud je základní tabulka indexově orientovaná a chceme uložit identifikátory ROWID základní tabulky do naší tabulky, měly by být ROWID, pokud je budeme testovat na shodu, uloženy ve sloupci UROWID. Jestliže jsou ROWID místo toho uloženy ve sloupci typu VARCHAR, pak může testování shody ROWID základní tabulky a ROWID naší tabulky v některých případech, přestože je vybrán stejný řádek, selhat, protože indexově orientované tabulky používají logické ROWID namísto fyzických. Logické ROWID mohou mít, na rozdíl od fyzických, pro stejný řádek různé textové reprezentace. Dva logické ROWID jsou shodné, pokud mají stejný primární klíč bez ohledu na adresu datového bloku v nich uloženou.

Sloupec UROWID může obsahovat jak fyzické, tak i logické ROWID. Uložení ROWID pro indexově orientovanou tabulku ve sloupci UROWID zajistí, že test na shodu vrátí vždy správný výsledek, a to i v případě, kdy dva logické ROWID mají stejný primární klíč, ale různé adresy datového bloku.

Pokud vytváříme tabulku pro uložení indexů se sloupcem ROWID z indexově orientované základní tabulky příkazem `CREATE TABLE AS SELECT`, pak se v naší indexové tabulce vytvoří sloupec UROWID o správné délce. Pokud ale nevytváříme tabulku se sloupcem ROWID z indexově orientované tabulky, musíme explicitně oznámit, že ROWID sloupec bude typu UROWID(x), kde x je velikost sloupce UROWID. Zvolená velikost by měla být dost velká, aby mohla obsáhnout jakékoliv ROWID ze základní tabulky, tudíž by to měla být funkce primárního klíče ze základní tabulky.

3.7.4 Metadata doménových indexů

Při použití klasických indexů v podobě B-stromu se uživatelé mohou pro získání informací o indexu dotazovat pohledu `USER_INDEXES`. Pro zajištění podobné podpory i pro doménové indexy můžeme poskytnout doménově specifická data, a to následujícím způsobem:

- Definovat jednu nebo více tabulek, které budou obsahovat tyto meta informace.
- Vytvoření pohledů, které spojí tabulky se systémově definovanými metadaty s indexovými meta tabulkami.

3.7.5 Pohledy doménových indexů

Jediné pohledy v souvislosti s doménovými indexy jsou pohledy poskytující informace o doménových indexech a o objektech s nimi spojených (tzv. sekundárních objektech). Sekundární objekty mohou být například tabulky vytvořené metodou `ODCIIndexCreate()` sloužící k ukládání indexových dat.

`ALL_SECONDARY_OBJECTS` – poskytuje informace o doménových indexech a jejich sekundárních objektech, které jsou přístupné uživateli.

`DBA_SECONDARY_OBJECTS` – poskytuje informace o všech doménových indexech a jejich sekundárních objektech v celé databázi.

`USER_SECONDARY_OBJECTS` – poskytuje informace o doménových indexech a jejich sekundárních objektech náležících danému uživateli.

Příklad použití pohledů sekundárních objektů pro získávání informací z `idx1` pomocí `USER_SECONDARY_OBJECTS`:

```
SELECT SECONDARY_OBJECT_OWNER, SECONDARY_OBJECT_NAME
FROM USER_SECONDARY_OBJECTS
WHERE INDEX_OWNER = userT AND INDEX_NAME = 'idx1'
```

Kód 3.12: Příklad zjišťování informací o doménovém indexu

Detailnější popis jednotlivých pohledů je dostupný v [16].

3.7.6 Rozdělené doménové indexy

Doménový index může být rozdělen (partitioned) stejně jako příslušná tabulka tak, aby každý jeho oddíl odpovídal oddílu tabulky, rozdělené podle rozsahu nebo výčtu hodnot. Takový index se nazývá **lokální (rozdělený) doménový index**. Lokální doménový index odkazuje na rozdělený index jako na celek, nikoliv na jednotlivé oddíly, a je rozdělen podle odpovídající tabulky tak, že všechny jeho klíče odkazují na řádky jen v dané části tabulky.

Naproti lokálním indexům existují i globální, a to rozdělené a nerozdělené. **Globální rozdělený index** je index rozdělené nebo nerozdělené tabulky, který je rozdělen jiným dělícím klíčem než příslušná tabulka. Takové indexy lze dělit pouze podle rozsahů. **Globální nerozdělený index** není rozdělen na oddíly a odpovídá indexu nerozdělené tabulky.

3.7.6.1 Systémové rozdělování

Systémové rozdělování umožňuje vytvoření jedné tabulky sestávající se z více fyzických oddílů. Při systémovém rozdělování se místo použití dělicího klíče (partitioning key) vytváří konkrétní počet oddílů. Proto nejsou výsledné oddíly ohraničeny, nemají žádné hodnoty, ani nejsou vytvořeny nějakou dělící metodou. Jelikož neexistuje žádný dělicí klíč, musí být sloupce rozdělované tabulky konkrétně mapovány do jednotlivých oddílů.

3.7.7 Systémově řízené doménové indexy

Použití systémově řízených doménových indexů má oproti uživatelsky řízeným doménovým indexům mnohé výhody. Následuje souhrn tří základních důvodů, proč používat systémově řízené doménové indexy.

- Protože jádro vykonává více úkolů údržby místo uživatele, není třeba pro údržbu tabulek a oddílů programová podpora. Tyto operace jsou implementovány serverem, proto vyžadují jen minimální programování ze strany uživatele. Kód cartridge pak většinou nemusí řešit problémy s oddíly.
- Počet objektů, které musí být spravovány pro podporu lokálních rozdělených doménových indexů, je stejný jako pro nerozdělené doménové indexy. Tabulky pro uložení doménových indexů jsou v případě lokálních rozdělených indexů rozděleny rovnoměrně s ohledem na základní tabulky, proto se počet tabulek s doménovými indexy s rostoucím množstvím oddílů nezvyšuje.
- K přístupu a manipulaci se systémově rozdělenými úložnými tabulkami stačí jeden dotaz a DML příkaz. Není již potřeba samostatná sada kurzorů pro každý oddíl, čímž se usnadňuje sdílení kurzoru a zlepšuje celkový výkon.

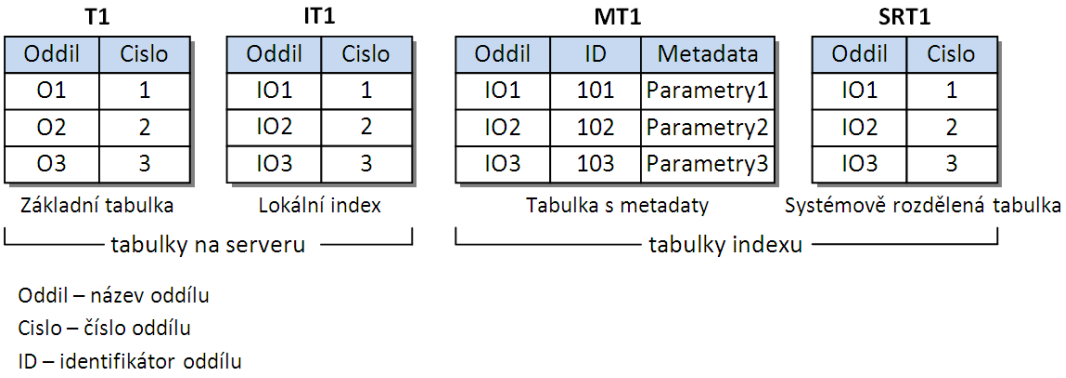
Nové verze Oracle Database již nebudou uživatelsky řízené doménové indexy podporovat, proto se doporučuje vyvíjet nové aplikace za pomoci systémově řízených doménových indexů.

Systémově řízený přístupu k doménovému indexování má následující omezení:

- Podporuje nerozdělené systémově řízené doménové indexy.
- Lokální systémově řízené doménové indexy podporuje pouze u tabulek rozdělených podle rozsahu nebo výčtu hodnot.
- Systémově řízený doménový index může indexovat pouze jeden sloupec.
- Nemůžeme vytvořit bitmapový nebo unikátní doménový index.

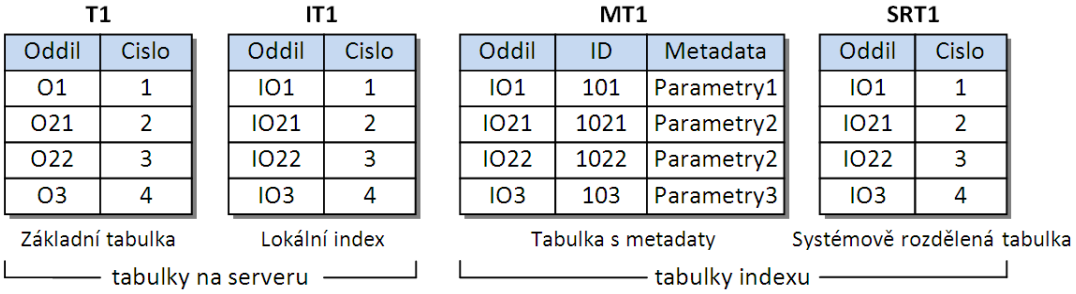
Na *Obr. 3.2* je základní tabulka ***T1*** s třemi oddíly, lokální doménový index ***IT1*** na jednom jejím sloupci, tabulka s odpovídajícími metadaty ***MT1***, což je volitelná tabulka vytvořená indexovým typem k uložení informací, specifických pro jednotlivé oddíly lokálního doménového indexu a čtvrtá tabulka je systémově rozdělená tabulka ***SRT1***, vytvořená indexovým typem k uložení indexových dat.

Tabulka *T1*, index *IT1* a tabulka *SRT1* mají všechny stejný počet oddílů v poměru 1:1. Tabulka *MT1* má tolik řádků, jako je počet oddílů předchozích tabulek.



Obr. 3.2: Rozdělené tabulky s doménovým indexem a související struktury

Nyní použijeme na toto rozdělení tabulek příkaz `ALTER TABLE T1 SPLIT PARTITION O2 INTO O21, O22` (viz *Obr. 3.3*). Ten způsobí, že se oddíl *O2* základní tabulky *T1* rozdělí na 2 oddíly, a to *O21* a *O22*. V lokálním doménovém indexu je oddíl *IO2* smazán a vytvoří se místo něj dva nové oddíly – *IO21* a *IO22*. Indexový typ zavolá metodu `ODCIIndexUpdPartMetadata()`, která aktualizuje tabulku *MT1*. Dále se v systémově rozdělené tabulce *SRT1* smaže oddíl *IO2* a vytvoří se místo něj dva nové. A nakonec jsou indexové oddíly označeny jako `UNUSABLE`, pro změnu na `USABLE` musí být přebudovány.



Obr. 3.3: Změna tabulek po provedení ALTER

Konkrétní ODCI metody používané pro systémově řízené doménové indexy a jejich popis lze nalézt v [16].

3.8 Vytváření objektů v doménovém indexování

Objekty je třeba vytvářet v určitém pořadí podle jejich závislostí, a to:

1. **Funkce, balíčky a objektové typy** – jsou odkazovány operátory a indexovými typy.
2. **Operátory** – jsou odkazovány indexovými typy a DML dotazy.
3. **Indexové typy** – jsou odkazovány doménovými indexy.
4. **Doménové indexy** – jsou používány DML dotazy.

Jak již bylo naznačeno v předchozích kapitolách, pro chování objektů při jejich mazání jsou možné dvě volby, a to buď **`RESTRICT`**, nebo **`FORCE`**. `RESTRICT` znamená, že pokud existují nějaké objekty závislé na mazaném objektu, je vymazání zakázáno. Defaultně mají toto chování nastavené funkce a balíčky. Naopak v případě volby `FORCE` je objekt smazán i v případě, že jsou na něm jiné objekty závislé a jsou zrušeny veškeré jejich vazby. Toto chování mají nastaveny objektové typy, operátory a indexové typy.

4 Indexování pohybujících se objektů

Jednou z mnoha oblastí vhodných pro použití doménových indexů jsou databáze pohybujících se objektů a s nimi spojené indexování časoprostorových dat. Vzhledem k neustále narůstajícímu objemu těchto dat má také stále větší význam zabývat se, jak k nim co nejefektivněji přistupovat. Pro tyto struktury neexistuje v databázi Oracle přímá podpora, proto by bylo užitečné naimplementovat pro tuto doménu index, který by takováto data uměl efektivně zpracovávat. K tomu je však zapotřebí nejdříve vybrat konkrétní strukturu a detailně se s ní seznámit, aby mohl být doménový index odpovídajícím způsobem naimplementován. V tomto ohledu je také bezpodmínečně nutné zavést si základní pojmy z oblasti zpracovávání časoprostorových dat.

Základním cílem databází pohybujících se objektů je rozšíření databázových systémů o podporu pro ukládání, reprezentaci dat a jejich dotazování. Obecně můžeme pohybující se objekty a jejich časoprostorová data rozdělit do dvou skupin, a to na objekty, u kterých nás zajímá jejich současná nebo budoucí pozice a na data, u kterých pracujeme s tzv. historickými daty - daty, která zaznamenávají události v minulosti. Historická data můžeme navíc zpracovávat a přistupovat k nim buď jako k diskrétním hodnotám měnícím se pouze jednou za určitý čas, nebo jako k hodnotám měnícím se spojitě.

Zjednodušeně se dá říci, že pohybující se objekty jsou geometrické útvary (body, křivky, plochy, tělesa) měnící se s časem a trajektorie je popisem jejich pohybu. Reálný geografický prostor, ve kterém se objekty pohybují, je spojitý, takže i jejich pohyb musí být popisován jako určitá spojitá změna pozice v čase. Časovou složku pohybu můžeme zjišťovat pouze porovnáváním dvou různých hodnot, takže trajektorii definujeme jako záznam prostorových hodnot měnících se s časem.

Nejdříve bude vhodné objasnit si, jak chápat objekt jako takový. Jedná se o identifikovatelný prvek v reálném světě, určitým způsobem oddělitelný od ostatních objektů. Příkladem takového objektu může být člověk, auto apod. Objekt ale začne mít svůj význam pro účely databází až v případě, kdy zaznamenáme jeho trajektorii, a proto bude v následujícím popisu termín pohybující se objekt chápán jako objekt, ke kterému přiřazujeme trajektorii.

Z aplikačního pohledu je trajektorie záznam pohybu nějakého objektu, tj. záznam jeho pozic v určitých časových momentech, takže abychom získali trajektorii jako spojitou čáru, musíme ji nejdříve z těchto záznamů pozic vytvořit. Při hledání nejvhodnějších tvarů trajektorie spojující jednotlivé body využíváme různé druhy interpolací (viz dále). Při použití jakékoliv interpolace je ale třeba vzít v potaz, že výsledná křivka bude vždy jen jakýsi odhad pravděpodobného tvaru reálné trajektorie.

Objekty jsou do určité míry omezovány v pohybu okolní infrastrukturou, reprezentovanou například budovami, geografií, dopravní situací apod. Podle typu omezení rozlišujeme pohyb objektů na *volný* (např. hráči na hřišti, vzorky znečištění v jezeře), *omezený* (např. auta v dopravě, lidé

v budovách) a na *pohyb v síti*, což je speciální případ omezeného pohybu, kde jsou pozice vyhodnocovány vůči omezujícímu prostoru představovaného určitou sítí. I samotná infrastruktura může během pohybu objektu měnit svůj tvar nebo polohu (ulice je zablokovaná, doprava se pohybuje pomalu, atp.).

4.1 Trajektorie

Obecně mohou být trajektorie rozděleny do dvou skupin podle vlastností prostorových objektů. První skupinou jsou bezrozměrné objekty reprezentované pohybujícími se body a druhou jsou objekty, které mají určité rozměry, reprezentované pohybujícími se oblastmi. V případě oblastí se mohou kromě pozice měnit v čase i jejich rozměry. V této práci se budu soustředit na první skupinu, tedy na indexování pohybujících se bodů.

4.1.1 Definice trajektorie

Trajektorii T definujeme jako kontinuální mapování z časové $I \subseteq \mathbb{R}$ do prostorové $\mathbb{R}^2$ (2D plocha) domény takto:

$$I \subseteq \mathbb{R} \to \mathbb{R}^2 : t \mapsto \alpha(t) = \left(\alpha_x(t), \alpha_y(t)\right) \tag{4.1}$$

a

$$T = \left\{\left(\alpha_x(t), \alpha_y(t), t\right) | t \in I\right\} \subset \mathbb{R}^2 \times \mathbb{R}\,. \tag{4.2}$$

(převzato z [6])

4.1.2 Aplikační pohled na trajektorie

Jak již bylo popsáno, z aplikačního pohledu je trajektorie záznam pohybu objektu, přesněji záznamy jeho pozic ve specifických časech. Trajektorie jsou tedy většinou definovány jako sekvence trojic tří rozměrů (x, y, t):

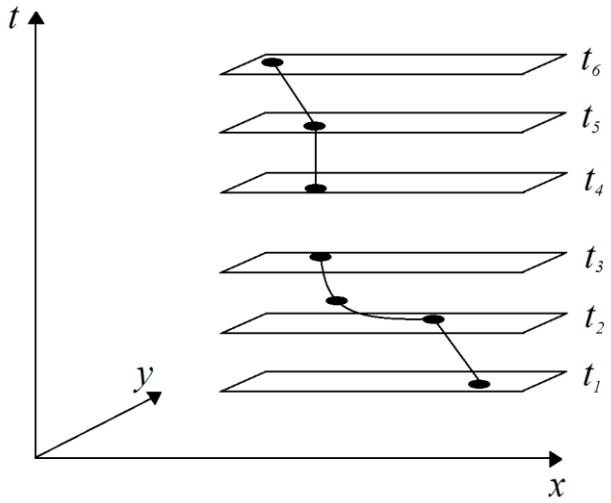
$$T = \{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n)\}, \tag{4.3}$$

kde $x_i, y_i, t_i \in \mathbb{R}$ a $t_1 < t_2 < \ldots < t_n$ a aktuální křivka trajektorie je aproximována použitím časoprostorových interpolačních metod na sadu trojic bodů daného objektu.

Dalším používaným pojmem bude segment trajektorie, který budeme chápat jako dvojici sousedících trojic (x, y, t):

$$S = \{(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})\}, \tag{4.4}$$

kde $x_i, y_i, t_i \in \mathbb{R}$ a $t_i < t_{i+1}$



Obr. 4.1: Trajektorie pohybujícího se objektu s interpolacemi

Nejjednodušší a nejrychlejší je *lineární interpolace*, která spojuje body pomocí přímky (na *Obr. 4.1* jde o segment mezi plochami t_1-t_2, t_4-t_5 a t_5-t_6). Interpolací pomocí *beziérových křivek* získáme mnohem hladší výslednou křivku (t_2-t_3). V každém případě samozřejmě platí, že čím kratší jsou časové vzdálenosti mezi jednotlivými body, tím dává použitá interpolace přesnější výsledky. Podrobnosti o interpolacích trajektorií lze také nalézt v [6].

4.2 Dotazování nad pohybujícími se objekty

Při dotazování nad trajektoriemi objektů je klíčové vyhledávání podle určitého rozsahu, které je odvozeno z prostorových a temporálních databází. Konkrétně se jedná o dotazy typu: „*Najdi všechny objekty v určité oblasti, resp. bodu, v určitém časovém rozsahu, resp. časovém okamžiku*“ nebo „*Najdi x nejbližších objektů k danému bodu v daném čase*“. Další druh používaného dotazu je tzv. time-slice dotaz, který v časoprostorovém kontextu najde pozice všech objektů v daném časovém bodě v minulosti. Při použití trojrozměrného modelu trajektorií prezentovaného výše, je time-slice dotaz speciálním případem rozsahového dotazu s rozsahovým oknem o nulové změně časové souřadnice.

Nová struktura dat nevyužívá jen stávající typy dotazů, ale přináší s sebou i nové způsoby dotazování, tzv. dotazy založené na trajektorii. Takové dotazy mohou být buď **topologické**, které se dotazují na obecné vlastnosti pohybu nebo **navigační**, které zjišťují z pohybu konkrétní informace, jako jsou například směr nebo rychlost.

Souhrnně tedy můžeme oddělit dvě skupiny dotazů, a to:

- **Dotazy založené na souřadnicích** (angl. coordinate-based) – dotazy na bod, rozsah a nejbližší sousedy ve trojrozměrném prostoru.
- **Dotazy založené na trajektorii** (angl. trajectory-based) – zahrnující výše zmíněné topologické a navigační dotazy.

Následně budou jednotlivé typy dotazů popsány konkrétněji a souhrnný přehled časoprostorových dotazů je uveden v *Tab. 4.1,* kde jsou uvedeny konkrétní operace a využívané typy. Složené závorky značí, že se jedná o soubor prvků daného typu.

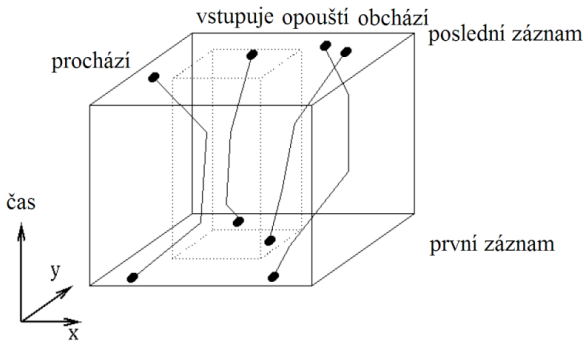
Druh dotazu		**Operace**	**Signatura operace**
Dotazy založené na souřadnicích		překrývá, uvnitř, atd.	časoprostorový rozsah × {segmenty} → segmenty
Dotazy založené na trajektoriích	Topologické dotazy	vstupuje, opouští, prochází, obchází	časoprostorový rozsah × {segmenty} → segmenty
	Navigační dotazy	uražená vzdálenost, pokrytá oblast, maximální nebo průměrná rychlost, směr	{segmenty} → integer {segmenty} → real {segmenty} → boolean

Tab. 4.1: Typy časoprostorových dotazů

4.2.1 Topologické dotazy

Tyto dotazy zahrnují celou nebo část trajektorie pohybujících se objektů. Jedná se o velmi důležité dotazy, ovšem s vysokou náročností na dobu zpracování. Z [3] je známý *9-intersection* model pro prostorová data, pro časoprostorová data však zatím není podobný model k dispozici. V [4] byl však představen soubor osmi základních rozšiřujících predikátů vycházejících z *9-intersection* modelu: *shoda, disjunkce, dotek, překrývá, obsahuje, pokrývá, je pokryto, uvnitř* a z nich odvozené další čtyři kompozitní: *vstupuje, opouští, prochází* a *obchází.*

Zjišťování těchto čtyř kompozitních predikátů je možné pouze vyhodnocením více než jen jednoho segmentu dané trajektorie. Pokud objekt například vstupuje do nějaké oblasti v určitém čase, pak musí platit, že počáteční bod jeho nejstaršího segmentu (resp. úsečky) je vně dané oblasti a zároveň koncový bod jeho nejnovějšího segmentu uvnitř této oblasti. Analogické uvažování uplatňujeme i v případě dalších tří predikátů. Všechny čtyři predikáty jsou znázorněny na následujícím obrázku (*Obr. 4.2*).



Obr. 4.2: Operace v topologických dotazech (převzato z [10])

4.2.2 Navigační dotazy

Informace o vlastnostech pohybu nejsou obvykle explicitně ukládány, ale mohou být odvozeny z uložených informací o trajektorii. Například průměrná nebo nejvyšší rychlost je určená ujetou vzdáleností za nějaký čas, směr pohybu je určen vektorem mez dvěma pozicemi nebo oblast je vypočítána pomocí konvexní obálky trajektorie. Všechny tyto vlastnosti mají společné to, že jejich hodnota vždy závisí na zadaném časovém intervalu a můžou se v závislosti na něm výrazně měnit.

V běžné praxi jsou dotazy zahrnující rychlost nebo směr velmi významné. Zde si můžeme uvést několik příkladů: „*Jakou rychlostí se vlak pohybuje?*“, „*Jaká byla maximální rychlost letadla?*“, „*Jakou vzdálenost ujelo auto dnes od 19.00 do 21.00?*“ apod. Ať už je časem v dotazu myšleno „teď“, nebo se uvažuje o větším časovém horizontu, v obou případech je třeba vyhodnotit několik segmentů dané trajektorie.

4.2.3 Kombinované dotazy

V těchto dotazech extrahujeme informace z trajektorií, resp. jejich částí tak, že nejdříve vybereme požadované trajektorie a následně jejich konkrétní části. Trajektorii je možné vybrat dotázáním se na její identifikátor (např. autobus číslo 45), vybráním segmentů pomocí časoprostorového rozsahu (autobus ve čtvrti Vysočany, dnes od 12 do 14 hodin), použitím topologického dotazu nebo i použitím nějaké odvozené informace.

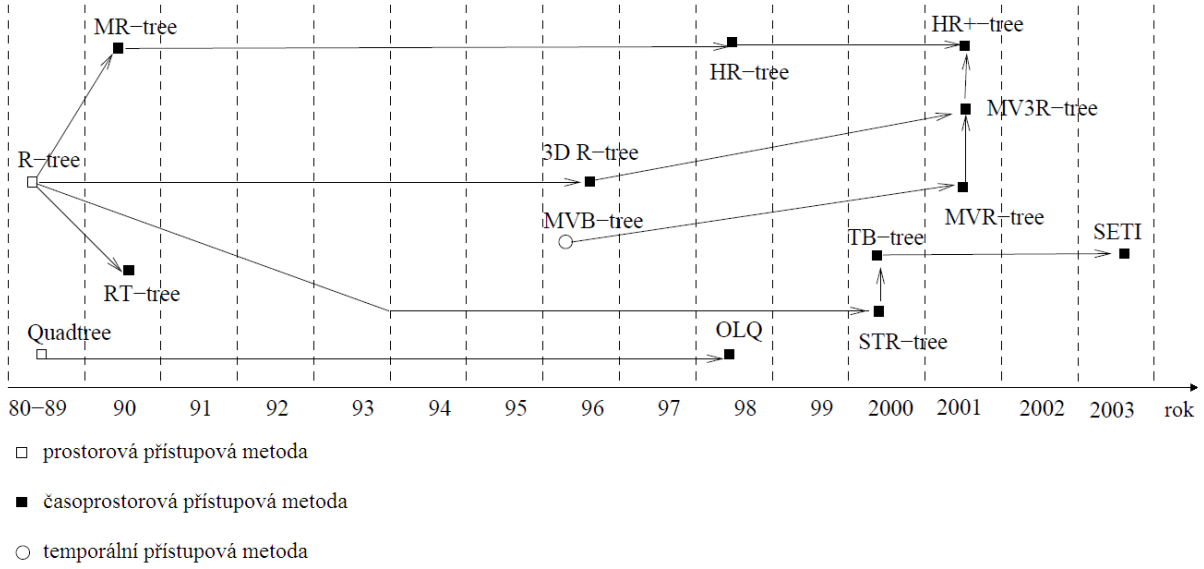
I pro kombinované dotazy si uvedeme několik příkladů:

„*Jaká byla vzdálenost ujetá autobusem číslo 45 včera?*“. Jako identifikátor je použito číslo autobusu.

„*Jaké byly trajektorie autobusů, které opustily Vysočany dnes v 10 hodin západním směrem, následující hodinu?*“. Zde je trajektorie identifikována časoprostorovým rozsahem a odvozenou informací, kterou je směr opuštění prostoru a délka vrácené trajektorie je omezena hodinou.

4.3 Přístupové metody

Existuje nepřeberné množství metod, kterými lze efektivně přistupovat k časoprostorovým datům. Klíčovým aspektem těchto přístupových metod je, zda indexují data historická, současná nebo budoucí. Dále se některé struktury mohou více zaměřovat na zpracování prostorových dat, některá více na temporální data, případně na obojí rovnoměrně. Pro tuto práci bude však důležitá oblast indexování historických dat, proto je na *Obr. 4.3* uveden výčet základních metod zaměřených na historická data.

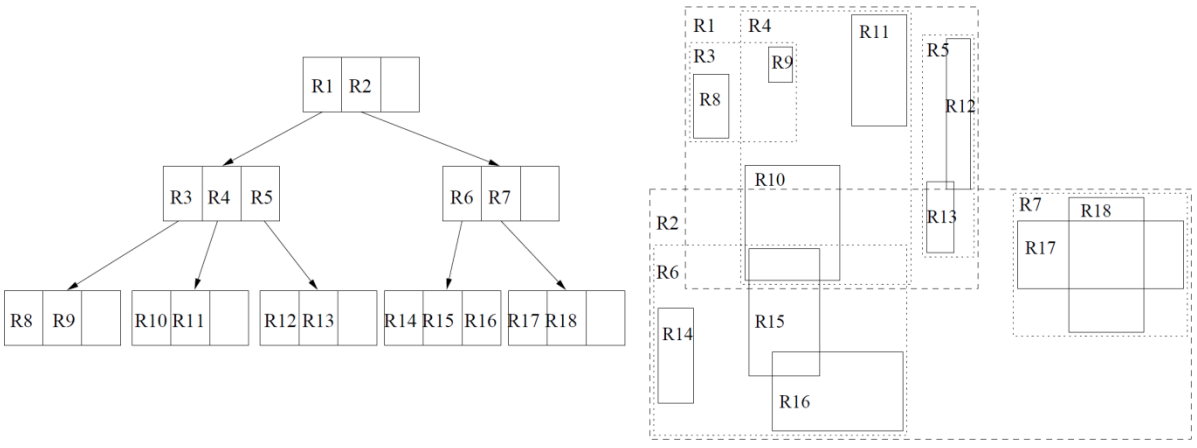


Obr. 4.3: Nejpoužívanější struktury pro indexování časoprostorových dat (částečně převzato z [9] a z [14])

V [9] a [14] je možné nalézt i další struktury pro indexování časoprostorových dat.

Výchozí strukturou pro přístup k časoprostorovým objektům je **R-strom**, proto nám jeho popis dá důležité podklady pro prezentaci nových, z něj odvozených, struktur.

R-strom je výškově vyvážený strom s indexovými záznamy v listech. Tyto záznamy obsahují ukazatele na konkrétní datové objekty. Konkrétně mají záznamy v listech tvar (*id*, *MBB*), kde id je identifikátor ukazující na aktuální objekt (v tomto případě segment trajektorie) a *MBB* (Mininal Bounding Box) je n-rozměrný interval. Nelistové záznamy mají tvar (*ptr*, *MBB*), kde *ptr* je ukazatel na potomka uzlu a *MBB* je podobně jako v předchozím případě n-rozměrný interval, tentokráte objímající celého potomka. Uzel ve stromu odpovídá stránce na disku. Minimální počet záznamů v každém uzlu je *m*, maximální počet pak *M*. Dolní hranice *m* brání před degenerací stromů tím, že pokud počet záznamů klesne pod tuto hranici, je uzel smazán a jeho záznamy jsou do stromu opětovně vloženy. Horní hranice *M* (tzv. fanout - rozvětvení) je určena velikostí stránky. Kdykoliv počet záznamů zvedne nad *M*, je uzel rozdělen.

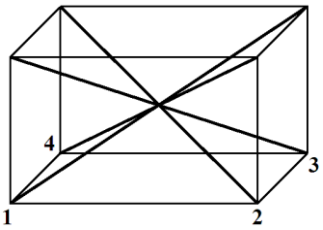


Obr. 4.4: Prostorová data a jejich uložení ve struktuře R-stromu

Při vkládání nového záznamu do R-stromu se projde strom od kořene až k listu. Tato cesta je vybrána pomocí tzv. kritéria nejmenšího rozšíření algoritmem `ChooseLeaf` , záznam je následně vložen a obalující *MBB* jsou podle změn upraveny algoritmem `AdjustTree`. V případě, že vložení způsobí rozdělení uzlu, pak jsou jeho záznamy přerozděleny do stávajícího a nově vytvořeného uzlu jedním z algoritmů `ExhaustiveSplit`, `LinearSplit` nebo `QuadraticSplit`. (Všechny výše zmíněné algoritmy jsou k nalezení v [8], kde byl R-strom poprvé představen.) Při mazání záznamu z R-stromu se uplatňuje opačný postup. Záznam se vymaže, *MBB* jsou podle toho následně upraveny, a pokud počet záznamů v uzlu klesne po hranici *m*, je smazán celý uzel a záznamy jsou znovu vloženy do stromu.

Při prohledávání R-stromu jej procházíme od kořene a zjišťujeme, zda se konkrétní záznam překrývá s vyhledávaným rozsahem. Pokud ano, vstoupíme do odpovídajícího potomka, ve kterém rekurzivně aplikujeme stejné kroky jako v rodičovském uzlu, a to do té doby, než získáme záznam ve vyhledávaném rozsahu. V případě R-stromu je možné, aby se jednotlivé uzly překrývaly, proto může být na každé úrovni stromu nalezeno více listů, ve kterých se objekt nachází (viz *Obr. 4.4*).

Z pohledu časoprostorových dat je ovšem tato technika poměrně neefektivní, protože vytváření *MBB* kolem všech úseček vede k velkému objemu nevyužitého prostoru. Je patrné, že *MBB* překrývají velké prostory, zatímco místo obsazené trajektorií samotnou je proti tomu velmi malé. To následně vede k nadměrnému překrývání a následkem toho k malým rozlišovacím schopnostem indexové struktury. Další nevýhodou R-stromů je, že nezaznamenávají, ke které trajektorii daný segment (úsečka) patří. Pro zlepšení výkonu R-stromu by tedy bylo vhodné přidat do záznamu v listu orientaci - údaj o směru úsečky v *MBB*, který může nabývat jednu ze 4 hodnot {1,2,3,4} (viz *Obr. 4.5*) a číslo trajektorie.



Obr. 4.5: Čtyři možnosti orientace úsečky v MBB

Aby se předešlo těmto problémům a následně se zrychlila práce s časoprostorovými daty, vznikly dvě nové struktury – **STR-strom** a **TB-strom**, popsané v [10]. Tato práce by měla být zaměřená na implementaci TB-stromu, pusťme se tedy do popisu právě této struktury.

4.3.1 TB-strom

Základní předpoklad při použití R-stromu je ten, že všechny vkládané útvary (v našem případě úsečky) jsou na sobě nezávislé. Nicméně úsečky jsou součástí trajektorií a tato znalost je jediná v tomto směru, která je implicitně v R-stromu i STR-stromu obsažena. TB-strom je, podobně jako

R-strom, výškově vyvážený strom s indexovými záznamy v listech. TB-strom se zaměřuje na striktní zachovávání trajektorií takovým způsobem, že jeden listový uzel obsahuje pouze segmenty jedné trajektorie. Odtud plyne název TB - Trajectory Bundle (svazek trajektorií). Tento přístup s sebou však nutně přináší určitá omezení buď v překrývání uzlů, nebo v prostorové diskriminaci (jedná se o jakési rozlišení indexu, čím vyšší prostorová diskriminace je, tím lépe zjišťujeme polohu jednotlivých objektů). Úsečky náležící k jiným trajektoriím, přestože leží blízko u sebe v prostoru, nejsou uloženy ve stejném uzlu. Se zvyšujícím se překrýváním prostorová diskriminace klesá, tím pádem cena rozsahových dotazů roste. Na druhé straně, pokud na prostorovou diskriminaci hledíme méně, získáme zase jiné výhody související se zachováváním trajektorií, proto je nutné učinit určité kompromisy.

Záznamy v listech R-stromu i STR-stromu mají tvar (*id_objektu*, *číslo_trajektorie*, *MBB*, *orientace*). Jelikož TB-strom nedovoluje, aby v jednom listu byly uloženy různé trajektorie, je *číslo_trajektorie*, spíše než do každého záznamu, přiřazeno do hlavičky listu. Formát záznamu v listu je tedy (*id_objektu*, *MBB*, *orientace*), přičemž *číslo_trajektorie* je uloženo pouze jednou.

4.3.1.1 Vkládání do TB-stromu

Cílem vkládání záznamů do stromu je rozdělit celou trajektorii pohybujícího se objektu na části, kde každá část má velikost *M* úseček (list obsahuje *M* segmentů trajektorie). Celý algoritmus je popsán pseudokódem (*Kód 4.1* a *Kód 4.2*), kde *N* značí uzel stromu (kořen), *E* vkládaný záznam, *E'* průběžně nalezený záznam a *N'* následníka nalezeného uzlu, na kterého odkazuje záznam *E'*. Na následujícím obrázku (*Obr. 4.6*) je znázorněn postup při vkládání záznamu.

```
Insert(N,E) {
  FindNode(N,E)
  IF uzel N' je nalezen THEN
    IF N' má místo THEN
      vlož do něj nový segment
    ELSE
      Vytvoř pro nový segment nový list
  ELSE
    Vytvoř pro nový segment nový list
}
```

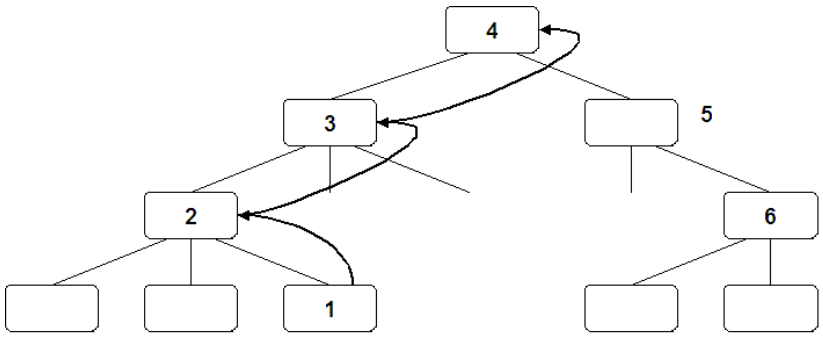
Kód 4.1: Algoritmus vkládání do TB-stromu

```
FindNode(N,E) {
  IF N není list THEN
    FOR(každý záznam E' uzlu N, jehož MBB se protíná s MBB
    E)
```

```
      FindNode(N',E)
  ELSIF N obsahuje záznam, který je napojený na vkládaný E
  dané trajektorie THEN
    RETURN N
}
```

Kód 4.2: Algoritmus vyhledání uzlu

Nyní si postupně pomocí *Obr. 4.6* popíšeme jednotlivé kroky vkládání. Abychom mohli vložit nový záznam, musíme nejdříve najít listový uzel, který obsahuje jeho předchůdce v konkrétní trajektorii. Začneme procházením stromu od kořene a vstoupíme do každého následníka, který se překrývá s *MBB* nové úsečky. Rekurzivně tak najdeme list obsahující segment, který je napojený na nový záznam (č. *1*). Postup nalezení segmentu je shrnut v algoritmu `FindNode` (*Kód 4.2*). V případě, že je list plný (počet záznamů dosáhne *M*), je potřeba uzly nějak rozdělit. Rozdělení by však narušilo stanovenou zásadu zachovávání trajektorií, takže namísto dělení pouze vytvoříme nový list. Poté procházíme stromem směrem vzhůru, dokud nenalezneme rodičovský uzel, ve kterém je místo (č. *2-4*) a vybereme volnou cestu pro vkládání (č. *5*). Pokud je v rodičovském uzlu místo (č. *6*), přiřadíme mu nový list. V případě, že je tento uzel plný, rozdělíme ho vytvořením nového nelistového uzlu a přiřadíme k němu náš nový list jako jeho jediného následníka. V případě potřeby propagujeme tyto změny výše. TB-strom roste směrem doprava, to znamená, že nejstarší, resp. první vložený list, je levý a naopak.

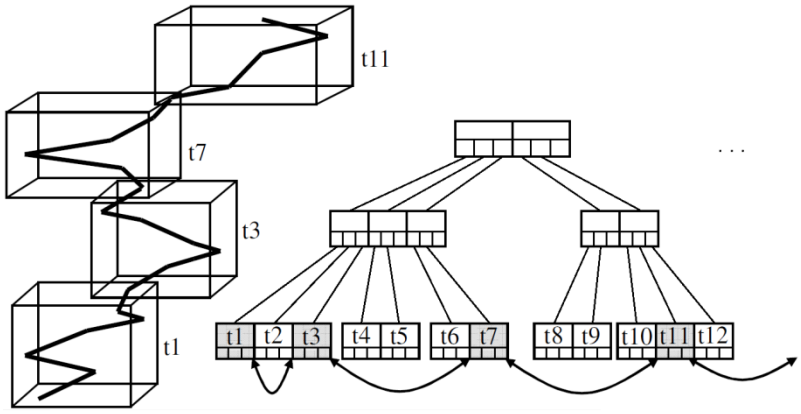


Obr. 4.6: Postup vkládání záznamu do TB-stromu

V případě, že bychom indexovali jakákoliv třírozměrná data, bez ohledu na to, jakých typů tyto rozměry jsou, a zcela tím zanedbávali důležitou vlastnost, kterou by měl prostorový index obsahovat - prostorovou diskriminaci, docházelo by k vysokému stupni překrývání. V případě TB-stromu však je prostorová diskriminace zohledněna, i když pouze v časovém rozměru, a to tak, že jsou data vkládána postupně s rostoucím časem (tzv. append-only).

Samotná struktura TB-stromu je soubor listových uzlů, kde každý obsahuje část trajektorie, organizované do stromové struktury. Takže trajektorie je rozdělená do více samostatných listů. Občas je nutné získat segmenty konkrétní dané trajektorie, proto bylo nutné související listy propojit nějakou dodatečnou strukturou. Jako nejvhodnější se ukázal zřetězený seznam, který propojuje listy obsahující

části stejné trajektorie. *Obr. 4.7* ukazuje část struktury TB-stromu a část příslušné trajektorie. Trajektorie je zde znázorněna tmavou čárou a prochází postupně čtyřmi listy (*t1, t3, t7, t11*,…), které jsou propojené zřetězeným seznamem.



Obr. 4.7: Struktura TB-stromu

Pokud se nacházíme v určitém listu, zřetězený seznam nám umožní získat celou trajektorii (nebo její část) s minimálními náklady. Pokud obsahují listy *r* záznamů, pak i délka (resp. počet segmentů) části trajektorie v listu je *r*. Předpokládáme-li, že *r* má hodnotu nejméně 3, nachází se v listu segmenty tří typů: první segment připojený pouze koncovou částí, následně *r-2* segmentů spojených s ostatními z obou stran a poslední segment připojený pouze svou počáteční částí. K nalezení dalších segmentů dané trajektorie tedy stačí procházet listy podle těchto ukazatelů.

4.3.2 STR-strom

STR-strom je rozšíření R-stromu s pozměněným algoritmem vkládání a dělení listů. Hlavní myšlenkou této struktury je snaha o dodržení prostorového uspořádání a zároveň alespoň částečné zachování trajektorie tím, že do určité míry udržuje segmenty stejné trajektorie blízko u sebe a zároveň zohledňuje jejich vzdálenost v prostoru, jako je tomu v případě R-stromů. Touto mírou je nově zavedený parametr *p* (preservation parameter), který se využívá právě k vyvažování mezi prostorovým uspořádáním a zachováním trajektorie.

Při vkládání nového segmentu je cílem vložit jej nejblíže jeho předchůdci v trajektorii, pouze však do výšky stromu *p*, jinak se postupuje obdobně jako u R-stromu. Menší p tedy snižuje míru zachování trajektorie, ale zvyšuje zohlednění prostorového uspořádání a naopak.

Podrobný popis STR-stromu lze nalézt v [10].

4.3.3 Algoritmy pro zpracování dotazů s použitím TB-stromu

Nyní si popíšeme způsoby zpracování dříve prezentovaných druhů dotazů v kontextu TB-stromu.

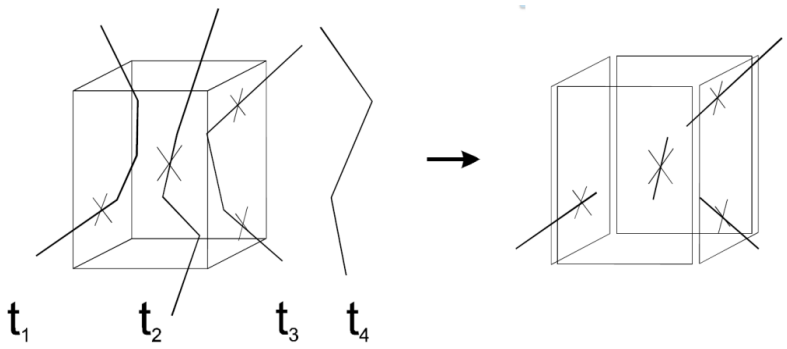
Zpracování dotazů založených na souřadnicích je přímé rozšíření klasických rozsahových dotazů zpracovávaných pomocí R-stromu. Principem je procházet strom s ohledem na zadané

souřadnice, dokud nenalezneme záznamy v jeho listech. TB-strom tento princip zachovává, proto se dále můžeme věnovat dotazům, které jsou v souvislosti s TB-stromem poněkud zajímavější, tedy topologickým a kombinovaným dotazům.

4.3.3.1 Topologické dotazy

Topologické dotazy jsou založeny na výše zmíněném *9-intersection* modelu a na jeho čtyřech kompozitních predikátech (*vstupuje*, *opouští*, *prochází* a *obchází*).

Zaměříme-li se na tento konkrétní dotaz: „*Které autobusy opustily Žďár nad Sázavou mezi dnešní 7 a 8 hodinou ranní?*“, vidíme, že cílem je vyhledat všechny autobusy, které opustily určitý časoprostorový rozsah. Na *Obr. 4.8* reprezentuje první kostka časoprostorový rozsah a t_1 je trajektorie pohybujícího se objektu, jenž do tohoto rozsahu vstupuje, trajektorie t_2 rozsah opouští, trajektorie t_3 jím prochází a trajektorie t_4 jej obchází. Abychom zjistili, který z těchto vztahů je mezi trajektorií a daným časoprostorovým rozsahem, musíme určit průniky trajektorie se všemi čtyřmi bočními stěnami zadaného rozsahu, resp. kostky. Jak je naznačeno na obrázku, pokud trajektorie do kostky vstupuje nebo ji opouští, pak nacházíme jediný průnik. V případě, že trajektorie do kostky vstupuje, musí platit, že počáteční bod segmentu pronikajícího se stěnou je vně této kostky. Pokud trajektorie kostku naopak opouští, musí být počáteční bod pronikajícího segmentu uvnitř kostky. Pokud najdeme průniků trajektorie s kostkou více, znamená to, že trajektorie kostku prochází. A nakonec v případě žádného průniku kostku obchází. Tímto způsobem můžeme ještě s pomocí rozsahových dotazů odpovídat na topologické dotazy.

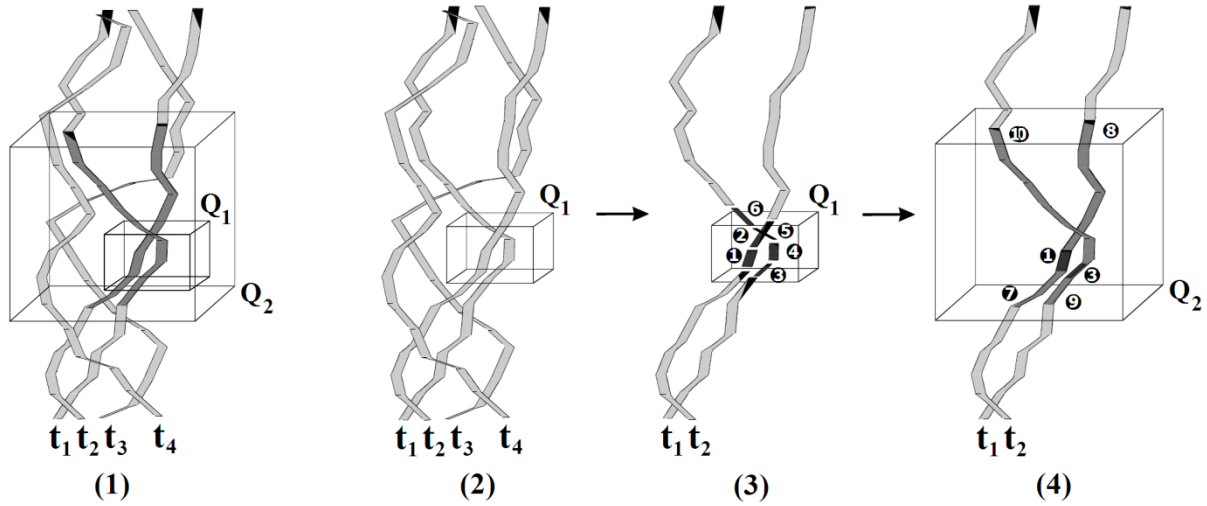


Obr. 4.8: Zpracování topologických dotazů (převzato z [21])

4.3.3.2 Kombinované dotazy

Prvním krokem vyhodnocení kombinovaného dotazu je získání základních dat v daném časoprostorovém rozsahu. K tomu můžeme použít například základní rozsahový dotaz používaný v R-stromu, projít strom a získat potřebné listy. Na *Obr. 4.9* jsou tyto kroky zobrazené pod čísly *(2)* a *(3)*. Rozsah je určený kostkou Q_1. Tímto dotazem získáme segmenty trajektorie t_2, označené čísly *1* a *2* a čtyři segmenty trajektorie t_1, označené čísly *3*, *4*, *5* a *6*. Získáním těchto segmentů končí první fáze kombinovaného dotazu.

V druhé fázi je třeba získat konkrétní části trajektorií, určených kostkou Q_2. Zde se projeví výhoda použití zřetězeného seznamu, který nám umožní bez hledání získat další spojené segmenty trajektorie. Ty se mohou nacházet buď ve stejném listu jako již nalezené segmenty, nebo je musíme hledat v jiných listech. V takovém případě musíme postupovat na související listy oběma směry pomocí zřetězeného seznamu. Nejdříve jedním a následně druhým směrem až po dotazovaný rozsah Q_2. Při kombinovaných dotazech je třeba dbát na to, aby jedna trajektorie nebyla vrácena vícekrát, proto si pro každou trajektorii uložíme její identifikátor a před získáním každé další zkontrolujeme, zda tato již nebyla nalezena.



Obr. 4.9: Jednotlivé kroky kombinovaného vyhledávání v TB-stromu (převzato z [21])

Algoritmus pro kombinované vyhledávání v TB-stromu je popsán následujícím pseudokódem (*Kód 4.3*), kde *N* značí uzel stromu (kořen), *E* záznam, *N'* následníka nalezeného uzlu, na který odkazuje záznam *E', rozsah1* odpovídá výše popsanému rozsahu Q_1 a *rozsah2* rozsahu Q_2:

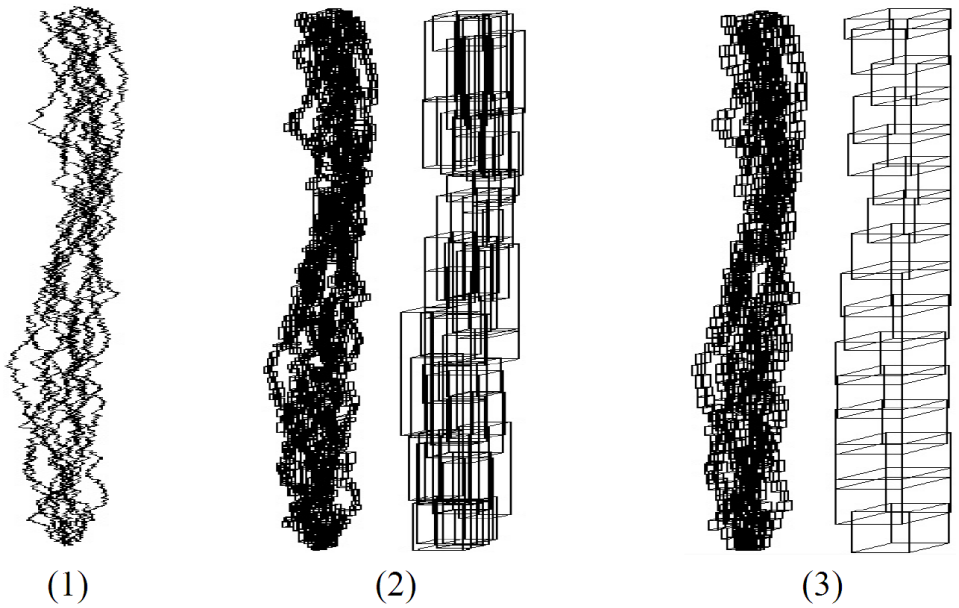
```
CombinedSearch(N,rozsah1,rozsah2) {
  IF N není list THEN
    FOR (každý záznam E' jehož MBB se protíná s rozsah1)
    CombinedSearch(N',E)
  ELSE
    FOR (všechny záznamy E, které odpovídají rozsah1 a
    jejichž trajektorie doposud nebyla získána)
      GetTrajectory(N,E)
}
GetTrajectory(N,E,rozsah2) {
  procházej N a hledej segment, který je napojený na konec
  E a označ jej jako E'
    WHILE E' je nalezeno AND E' je v rozsah2
```

```
      přidej E' k výsledku,
      procházej N a hledej segment, který je napojen na E'
      a označ jej jako E'
    IF E' není v uzlu nalezeno THEN
      nastav N na další podle ukazatele,
      opakuj celou proceduru od začátku
  proveď ještě jednou to samé, jen hledej segment napojený
  na začátek
}
```

Kód 4.3: Kombinované vyhledávání v TB-stromu

4.3.4 Porovnání prezentovaných struktur

Obr. 4.10 ukazuje sadu deseti trajektorií indexovaných R-stromem (*2*) a TB-stromem (*3*). Oba stromy mají výšku 3. Na obrázku jsou znázorněny první dvě úrovně každého stromu a jsou zde dobře patrné odlišnosti obou struktur. Významnější rozdíl je na druhé úrovni (na obrázku u jednotlivých bodů vpravo), kde je vidět vysoký stupeň překrývání R-stromu oproti TB-stromu.



Obr. 4.10: Trajektorie a MBB jednotlivých stromů: (1): trajektorie; (2): MBB R-stromu (úroveň 1, úroveň 2); (3): MBB TB-stromu (úroveň 1, úroveň 2); (částečně převzato z [21])

Z této malé prostorové diskriminace R-stromu oproti TB-stromu vyplývá i mnohem větší velikost indexové struktury. V tomto případě je dokonce téměř dvojnásobná. Studie v [10] ukazuje, že index souboru 1 000 pohybujících se objektů s celkovým počtem 1 500 000 segmentů má v podobě R-stromu velikost 95 MB, zatímco TB-strom 57 MB.

Jak je z dalších testů patrné [10], TB-strom zpracovává dotazy založené na trajektorii v porovnání s R-stromem mnohem efektivněji. V oblasti rozsahových dotazů se výkonnost

TB-stromu a R-stromu vyrovnává. V kombinovaných dotazech pak výkonnost TB-stromu oproti ostatním metodám úměrně roste se zvyšujícím se počtem pohybujících se objektů.

4.3.4.1 Slabé stránky TB-stromu

Vedle neoddiskutovatelných východ TB-stromu především v dotazech založených na trajektorii má i své podstatné zápory. Hlavní z nich spočívá ve strategii vkládání. Jak vyplývá z výše popsaných principů, nová data jsou vždy vkládána na pravý konec stromu, to znamená, že výkonnost stromu je velmi závislá na pořadí vkládaných dat. Pokud budeme vkládat data v chronologickém pořadí, pak budou časově blízké trajektorie uloženy blízko u sebe i v indexu, pak žádný problém nevzniká. V reálném světě však tento předpoklad není vždy zaručen. Může například dojít k situaci, kdy mezitím co jsou trajektorie ostatních objektů vkládány, nemusí být určitý objekt v zaznamenávané oblasti a jeho pohyb se začne zaznamenávat až za určitou dobu. Tím může dojít k velkému překrývání v časovém rozměru a tím pádem i výraznému zhoršení výkonu indexu.

5 Implementace doménového indexu

V kapitole 3 byly popsány obecné principy a postupy při implementování doménových indexů, v kapitole 4 pak následně prostudována problematika indexování časoprostorových dat a konkrétní struktura TB-stromu, která bude jako výsledek této práce použita v doménovém indexu. Tím tedy máme všechny potřebné podklady k vlastní implementaci doménového indexu a jeho následného použití v praxi.

5.1 Jazyk PL/SQL

K implementaci doménového indexu je využit procedurální jazyk databáze Oracle PL/SQL. Základem jazyka PL/SQL je jeho bloková struktura, takže základní elementy jazyka, jako jsou funkce, procedury, balíčky a další, utváří logické bloky, v nichž jsou deklarace lokální a nejsou mimo blok dostupné.

Jak je patrné (*Kód 5.1*), PL/SQL blok má tři základní části: deklarační, kde se deklarují objekty daného bloku, příkazovou část, ve které je s deklarovanými objekty manipulováno a také část pro zpracování výjimek z příkazové části. Jediná povinná část je příkazová.

```
[DECALRE
  -- deklarace]
BEGIN
  -- příkazy
[EXCEPTION
  -- zpracování výjimek]
END;
```

Kód 5.1: Bloková struktura jazyka PL/SQL

5.1.1 Výhody jazyka

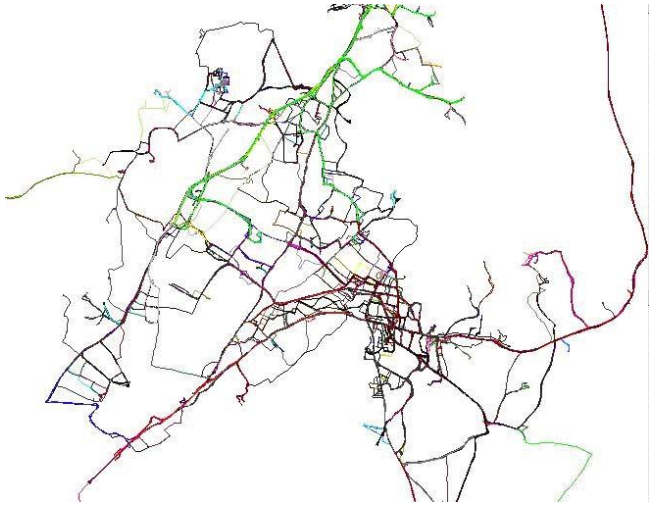
Jazyk PL/SQL je vhodný pro implementaci doménových indexů zejména z následujících důvodů:

- Je úzce integrován s jazykem SQL.
- Snižuje objem komunikace mezi aplikací a databází, tím nabízí lepší výkon.
- Je přenositelný.
- Podporuje objektově orientované programování.
- V tomto kontextu nabízí lepší produktivitu než jiné podporované jazyky.

Kompletní popis jazyka lze nalézt v dokumentaci Oracle [18].

5.2 Základní databáze

Zamýšlený doménový index by měl zefektivnit práci s trajektoriemi v naší databázi, která pracuje s trajektoriemi pohybujících se objektů, konkrétné nákladních vozidel. Pro naplnění této databáze byla použita reálná sada dat z [22] (viz *Obr. 5.1*). Ta konkrétně obsahuje 276 trajektorií 50 nákladních vozidel vozících beton na stavby v okolí řeckých Atén ve 33 dnech. Každý záznam obsahuje číslo objektu, číslo jeho trajektorie, souřadnice a čas, ve kterém se na daném místě nacházel.

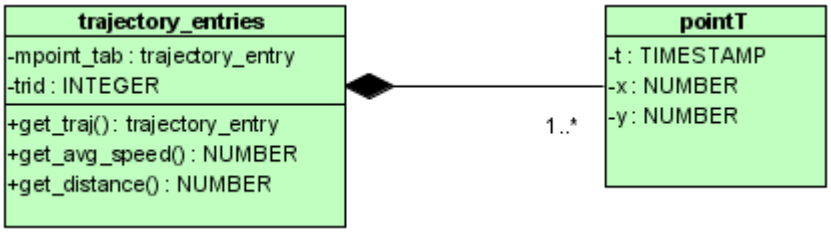


Obr. 5.1: Trajektorie nákladních vozidel (převzato z [22])

Při použití těchto dat bylo nutné provést jednoduché předzpracování, aby od sebe byly odděleny jednotlivé trajektorie. Všechna data jsou procházena a každé nové trajektorii, která je rozlišena změnou identifikace objektu nebo změnou časové kontinuity, je přiřazen identifikátor a je vložena do základní tabulky `trajTable`, nad jejímž sloupcem `trajectory` vytváříme index (viz dále).

5.3 Pohybující se bod

Vzhledem k tomu, že u námi indexovaných dat jsou rozměry objektu irelevantní, byl jako základní typ pohybujícího se objektu zvolen bezrozměrný bod. Při implementaci tohoto typu byla využita struktura nabízená systémem Oracle 11g, `NESTED TABLE` (vnořená tabulka). Ta nám umožňuje modelovat pohybující se bod jako kolekci záznamů o předem neznámém počtu a tzv. out-of-line ji uložit do jednoho sloupce tabulky, nad kterým následně vytvoříme index. Stejným způsobem lze ale použít i kolekci typu `VARRAY` (Variable Size Array) o dostatečné velikosti, aby obsáhla každou jednu vloženou trajektorii. Základní typ tedy vypadá následovně:



Obr. 5.2: Návrhové třídy základního indexovaného objektu

Na základě těchto typů byla vytvořena základní tabulka (*Kód 5.2*) a její sloupec `trajectory` typu `trajectory_entries` bude naplněn daty a následně indexován. `Objid` je identifikátor objektu, `trid` identifikátor jeho trajektorie a `trajectory` obsahuje trajektorii.

```
CREATE TABLE trajTable (
  objid INTEGER,
  trid INTEGER,
  trajectory trajectory_entries,
  CHECK (trajectory.trid = trid) )
NESTED TABLE trajectory.mpoint_tab STORE AS mpoint_nt;
```

Kód 5.2: Vytvoření základní tabulky

5.4 TB-strom

Pro každý ze základních elementů stromu byl vytvořen příslušný objektový typ. Vzhledem k odlišnosti listových a nelistových uzlů stromu bylo třeba tyto dva typy uzlů stromu rozlišit a pro každý namodelovat i jiný typ záznamů. Typy pro samotné záznamy jednotlivých segmentů již jsou společné. Základní struktura TB-stromu je uvedena na diagramu návrhových tříd (*Obr. 5.3*) a podrobně popsána v následující kapitole.

5.4.1 Implementační typy TB-stromu

TB-strom se stává ze dvou základních tříd, a to `leaf`, reprezentující listový uzel stromu, ve kterém jsou uloženy segmenty dané trajektorie, a `node`, reprezentující vnitřní nelistové uzly stromu.

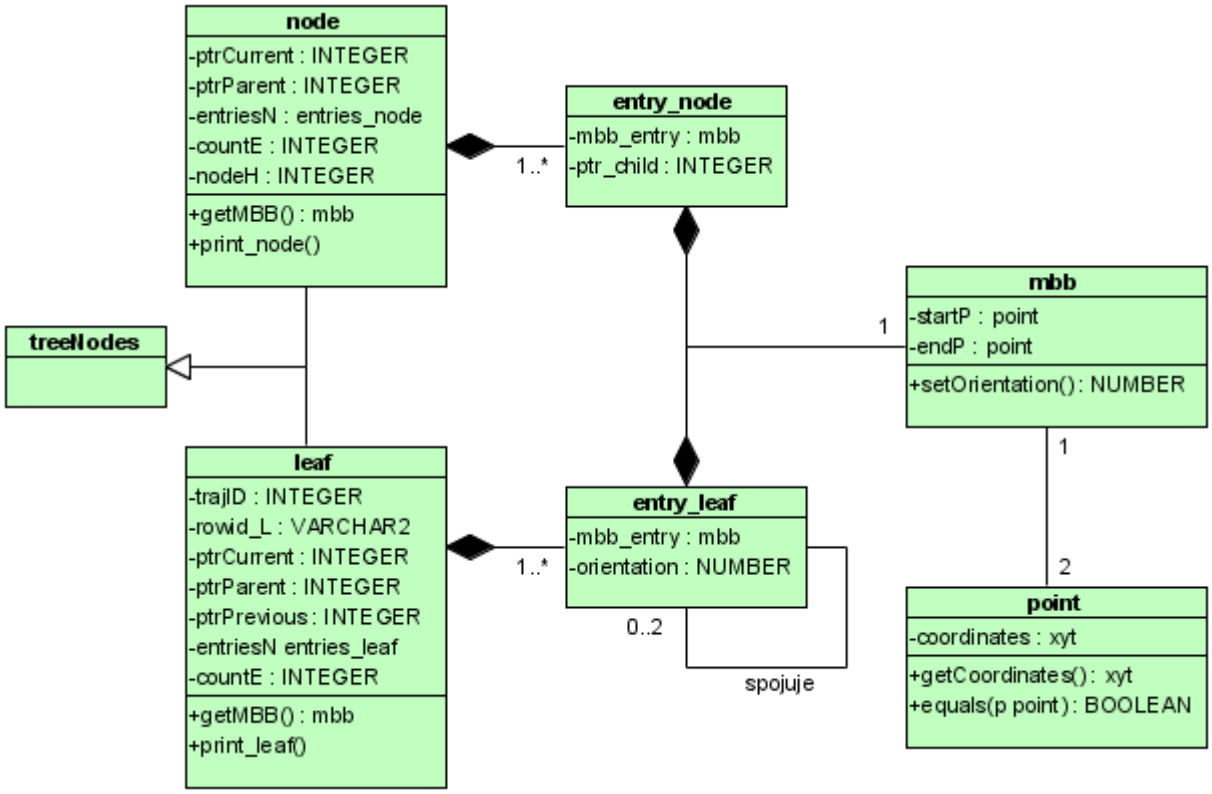
Prvním atributem třídy **`leaf`** je `trajID`, což je identifikátor jediné trajektorie v listu. Druhý atribut je `rowid_L`, který značí ROWID pohybujícího se objektu, jehož trajektorie je v listu obsažena. `PtrCurrent` je identifikátor daného uzlu a `ptrParent` je číslo ukazující na jeho rodičovský uzel. Dva atributy, pro list a TB-strom specifické, jsou `ptrPrevious` a `ptrNext`. Právě tyto dvě hodnoty se podílí na reprezentaci výše zmíněného zřetězeného seznamu, propojujícího listy stejné trajektorie. `PtrPrevious` (resp. `PtrNext`) je identifikátor předchozího (resp. následujícího) listu, obsahujícího segmenty stejné trajektorie. Pokud se jedná o první, poslední nebo jediný list dané trajektorie, má tento identifikátor stejnou hodnotu jako `ptrCurrent`, tedy list

v takovém případě ukazuje sám na sebe. Poslední atribut `countE` je čítač záznamů segmentů obsažených v listu.

Samotné záznamy se v listu skrývají pod atributem `entriesN`. Jedná se `VARRAY` o velikosti 55 segmentů trajektorie (viz níže), které jsou reprezentované třídou `entry_leaf`. Tato třída obsahuje záznam `mbb_entry` typu `mbb`, který obaluje daný segment, a číslo `orientation` značící jednu ze čtyř možných orientací segmentu v *MBB*. `Mbb_entry` je pak následně tvořen dvěma třírozměrnými body typu `point` – počátečním (`startP`) a koncovým (`endP`), které spoluutváří *MBB* segmentu.

Třída **`node`** je implementována na stejném principu jako třída `leaf`. Vzhledem k podstatě TB-stromu již ale nemusí ukládat identifikátor trajektorie a smysl v nelistovém uzlu ztrácí i proměnné pro zřetězený seznam. Naopak v nelistovém uzlu musí být, stejně jako v listu, uloženy atributy `ptrCurrent` (identifikátor daného nelistového uzlu) a `ptrParent`, ukazující na rodiče listu. Je-li uzel kořenem stromu, pak je hodnota `ptrParent` rovna nule. Uzel obsahuje pro implementační účely navíc proměnnou `nodeH`, která značí výšku daného uzlu ve stromové struktuře.

Samozřejmě i uzel typu `node` obsahuje atribut `entriesN`. Jako u listu, se i zde jedná o pole `VARRAY` velikosti 55, ovšem v tomto případě jsou tvořeny záznamy typu `entry_node`, které obsahují identifikátor následníka (`ptr_child`) a jeho *MBB* (`mbb_entry`). `Mbb_entry` je zde už opět stejného typu jako u listu a je tedy tvořeno dvěma trojrozměrnými body (počátečním a koncovým).



Obr. 5.3: Diagram návrhových tříd TB-stromu

Při určování maximálního počtu záznamů v uzlu se vycházelo z podmínky, že uzel nesmí překračovat velikost datového bloku, tedy 8 192 B. Stanovíme-li maximální velikosti čísel v hlavičce listu na 4 B a pole pro ukládání ROWID na 20 B, pak spočítáme velikost hlavičky listu, která může být maximálně 6*4+20, tedy 44 B. Maximální velikost jednoho záznamu v listu bude určena jedním číslem o maximální velikosti 4 B a šesti souřadnicemi o velikosti 24 B. Počet jednotlivých záznamů by pak měl být daný jako podíl velikosti bloku zmenšeného o velikost hlavičky listu, ku velikosti jednoho záznamu. Výsledný maximální počet záznamů v jednom listu je tedy stanoven na 55. U nelistových uzlů je hodnota maximálního počtu záznamů analogickým postupem stanovena také na 55. Každý existující listový nebo nelistový uzel musí obsahovat alespoň jeden záznam.

5.4.2 Funkce a procedury TB-stromu

5.4.2.1 InsertEntry

Procedura `InserEntry()` (*Kód 5.3*) je spuštěna vždy, když pohybující se objekt změní svou pozici.

```
CREATE PROCEDURE InsertEntry (sp IN point, objID IN
  INTEGER, row_id IN VARCHAR2, leafT IN VARCHAR2, nodeT IN
  VARCHAR2)
```

Kód 5.3: Signatura procedury InsertEntry()

Parametry této procedury jsou `sp` (nový bod dané trajektorie), `objID` (identifikátor objektu), `row_id` (ROWID trajektorie v základní tabulce), `leafT` a `nodeT` (názvy indexových tabulek).

Procedura `InserEntry()` vychází z výše popsaného algoritmu `Insert`. Tento algoritmus by měl pro vyhledání uzlu používat algoritmus `FindNode()`. Ten byl však z důvodu možnosti rychlejšího vkládání záznamů do indexu nahrazen vhodnou hashovací metodou. Využívá se zde struktury v paměti v podobě asociativního pole, které pro každou trajektorii spravuje dvojice záznamů ve tvaru: poslední záznam dané trajektorie (typu `point`) a ukazatel na uzel, kde se tento poslední záznam nachází. Místo časově náročného hledání ve stromu se pak jen podle čísla dané trajektorie vyhledá v asociativním poli její poslední záznam, a tím pádem uzel a konkrétní segment, za který se má nový záznam vložit.

`InsertEntry()` nejdříve výše popsaným způsobem nalezne odpovídající list, poté zjistí, zda je v listu místo. Pokud ano, vloží do něj záznam a procedurou `AdjustTree()` se změny propagují stromem až ke kořeni. Pokud v listu místo není, zavolá se funkce `CreateNewLeaf()`, která vytvoří a vrátí nový list. Pokud není žádný odpovídající list nalezen, pak se také funkcí `CreateNewLeaf()` vytvoří nový list. Nakonec se aktualizuje hodnota (resp. hodnoty) v hashovací struktuře.

5.4.2.2 CreateNewLeaf

Funkce `CreateNewLeaf()`, jak již bylo naznačeno, vytvoří nový list a vloží do něj první záznam.

```
CREATE FUNCTION CreateNewLeaf (objID IN INTEGER, row_id IN
  VARCHAR2, pntPrev IN point, pntCur IN point, leafT IN
  VARCHAR2, nodeT IN VARCHAR2) RETURN leaf
```

Kód 5.4: Signatura funkce CreateNewLeaf()

Jako argumenty přijímá `objID` (identifikátor objektu), `row_id` (ROWID trajektorie v základní tabulce), `pntPrev` (poslední již vložený bod dané trajektorie), `pntCur` (vkládaný bod dané trajektorie), `leafT` a `nodeT` (názvy indexových tabulek).

`CreateNewLeaf()` nejdříve zjistí, zdali je vytvářený list první dané trajektorie, nebo se bude muset napojit na nějaký existující list, podle toho se nastaví parametry zřetězeného seznamu. Poté je nový list vytvořen, je do něj vložen nový záznam vytvořený na základě `pntPrev` a `pntCur` a list je svázán s příslušným rodičem. Nakonec se provedené změny propagují procedurou `AdjustTree()` stromem vzhůru ke kořeni.

5.4.2.3 AdjustTree

`AdjustTree()` vychází z originálního algoritmu AdjustTree, prezentovaného v souvislosti s R-stromy [8].

```
CREATE PROCEDURE AdjustTree (old_leaf leaf, new_leaf leaf,
  node_tab VARCHAR2, leaf_tab VARCHAR2)
```

Kód 5.5: Signatura procedury AdjustTree()

Jejími parametry jsou kromě názvů indexových tabulek také změněný list `old_leaf`, případně nově vytvořený list `new_leaf`.

`AdjustTree()` závisí na tom, zda je vytvořen nový list, nebo ne. V případě, že je vložen nový záznam a nový list nemusel být vytvořen, pouze se projde strom až ke kořeni a patřičně se změní všechny související *MBB*. Pokud je však nový list vytvořen, musí se kromě toho ještě vložit do rodiče, kterého se změna týká, i nový záznam a v případě, že se záznam do uzlu nevejde, propagovat opět tuto změnu výše.

5.5 Implementační typ

Nyní jsou vytvořeny potřebné struktury pro implementaci tzv. implementačního typu, což je objektový typ definující metody doménového indexu. Implementační typ bude následně využíván indexovým typem, který musí být specifikován při samotném vytváření indexu nad tabulkou. Jak již bylo uvedeno výše, metody doménového indexu se dělí na metody pro definici, údržbu a

prohledávání indexu. Signatura vytvořeného implementačního typu je uvedena v následujícím kódu (*Kód 5.6*) a jednotlivé funkce a metody vysvětleny v dalších kapitolách. Jediná proměnná v implementačním typu je `curnum`, značící číslo kurzoru, který je využíván metodami pro prohledávání indexu.

```
CREATE TYPE DomIndex AS OBJECT
(
curnum NUMBER,
STATIC FUNCTION ODCIGetInterfaces(ifclist OUT
  sys.ODCIObjectList) RETURN NUMBER,
STATIC FUNCTION ODCIIndexCreate (ia sys.ODCIIndexInfo,
  parms VARCHAR2, env sys.ODCIEnv) RETURN NUMBER,
STATIC FUNCTION ODCIIndexDrop(ia sys.ODCIIndexInfo, env
  sys.ODCIEnv) RETURN NUMBER,
STATIC FUNCTION ODCIIndexInsert(ia sys.ODCIIndexInfo, rid
  VARCHAR2, newval trajectory_entries, env sys.ODCIEnv)
  RETURN NUMBER,
STATIC FUNCTION ODCIIndexUpdate(ia sys.ODCIIndexInfo, rid
  VARCHAR2, oldval trajectory_entries, newval
  trajectory_entries, env sys.ODCIEnv) RETURN NUMBER,
STATIC FUNCTION ODCIIndexStart(sctx IN OUT DomIndex, ia
  sys.ODCIIndexInfo, op sys.ODCIPredInfo, qi
  sys.ODCIQueryInfo, strt NUMBER, stop NUMBER, cmpval
  pointsS, cmpvalT pointsT, env sys.ODCIEnv) RETURN NUMBER,
MEMBER FUNCTION ODCIIndexFetch(self IN OUT DomIndex, nrows
  NUMBER, rids OUT sys.ODCIRidList, env sys.ODCIEnv) RETURN
  NUMBER,
MEMBER FUNCTION ODCIIndexClose(env sys.ODCIEnv) RETURN
  NUMBER
);
/
CREATE OR REPLACE TYPE BODY DomIndex IS
...
END;
/
```

Kód 5.6: Vytvoření implementačního typu

5.5.1 Metody pro definici indexu

Z metod pro definici indexu byly vytvořeny následující dvě. Metoda pro změnu indexu `ODCIIndexAlter()` nebyla z důvodu malé využitelnosti v kontextu prezentovaného indexovacího schématu implementována.

5.5.1.1 ODCIIndexCreate()

Tato metoda je volaná při vytváření indexu příkazem `INDEX CREATE` spolu s odkazem na implementovaný indexový typ.

Parametry této metody jsou:

- `ia` typu `ODCIIndexInfo` – obsahuje informace o indexu a indexovaném sloupci. Nejdůležitější atributy tohoto typu jsou mimo jiné `IndexSchema`, `IndexName` nebo `IndexCols`. Při vytváření indexu je například využívána hodnota `ia.IndexCols(1).Tablename`, která obsahuje název indexované tabulky, v hodnotě `ia.IndexSchema` je získán název schématu obsahujícího index a další, jejichž seznam lze nalézt v dokumentaci [16].
- `parms` – zde jsou předávány parametry, pokud je index s nějakými vytvářen.
- `env` typu `ODCIEnv` – obsahuje obecné informace o prostředí, ve kterém je metoda vykonávána.

`ODCIIndexCreate()` nejdříve vytvoří dvě objektové tabulky pro ukládání indexu. První tabulka **`IDX1_tree_leaf`** se využívá pro ukládání listů stromu, druhá tabulka **`IDX1_tree_node`** pro ukládání nelistových uzlů stromu. Dále je vytvořena tabulka `IDX1_mob` pro uložení hashovací struktury.

Dalším krokem je vytvoření kurzoru nad indexovanou tabulkou `trajTable`, a pokud tabulka obsahuje nějaké záznamy, pak procházení všech těchto záznamů a jejich postupné načítání do struktury TB-stromu výše popsanou procedurou `InsertEntry()`. Všechny tři tabulky jsou následně naplněny novými daty.

5.5.1.2 ODCIIndexDrop()

Tato metoda je volána příkazem `INDEX DROP` a slouží k vymazání doménového indexu. Provádí to jednoduše vymazáním všech dříve vytvořených indexových tabulek a všech dalších indexových struktur.

Parametry této metody jsou stejné jako u `ODCIIndexCreate()`, jen se zde nevyužívá parametr `parms`.

5.5.2 Metody pro údržbu indexu

5.5.2.1 ODCIIndexInsert()

Metoda `ODCIIndexInsert()` je volána při vložení nových záznamů nebo celé nové trajektorie do indexované tabulky příkazem `INSERT INTO`.

Dva parametry této metody – `ia` a `env` již byly popsány u dříve uvedených metod. Jsou zde však také využívány dva nové parametry, a to:

- `rid` – značí identifikátor nově vkládaného řádku v indexované tabulce.
- `newval` – v této hodnotě jsou předávány všechny záznamy nově vkládané trajektorie (typu `trajectory_entries`) spolu s číslem této trajektorie.

`ODCIIndexInsert()` nejdříve zjistí, zda je vkládaná trajektorie již ve stromu obsažena. Pokud tomu tak je, spustí se pro všechny její nové záznamy procedura `InsertEntry()`, do tabulek se vloží nově vytvořené záznamy a provede se potřebná aktualizace. V opačném případě `ODCIIndexInsert()` spustí proceduru `ODCIIndexUpdate()`, která provede aktualizaci hodnot v trajektorii.

5.5.2.2 ODCIIndexUpdate()

Tato metoda je v tomto případě volána při aktualizaci indexované tabulky pomocí `UPDATE` nebo metodou `ODCIIndexInsert()`.

Sada parametrů nabízí oproti `ODCIIndexInsert()` pouze jeden navíc, a to parametr `oldval` (stejného typu jako `newval`, tedy `trajectory_entries`). `Oldval` reprezentuje hodnotu v tabulce, kterou požadujeme nahradit novou hodnotou v `newval`.

`ODCIIndexUpdate()` zajišťuje vkládání segmentů do existujících trajektorií. I tato metoda využívá ke vkládání do TB-stromu proceduru `InsertEntry()` a také provádí související aktualizaci záznamů v indexových tabulkách.

5.5.2.3 ODCIIndexDelete()

Tato metoda by měla sloužit k vymazání záznamů z indexu při jejich vymazání ze základní tabulky. Vymazávání jednou zapsaných záznamů však není v kontextu naší databáze třeba, proto tato metoda není implementována.

5.5.3 Metody pro prohledávání indexu

Metody jsou implementovány modelem tzv. *předpočítání všeho* (viz *kap. 3.4.3*), kdy je v metodě `ODCIIndexStart()` vypočítána celá sada výsledků a několikanásobným voláním metody `ODCIIndexFetch()` následně vráceny konkrétní řádky základní indexované tabulky.

5.5.3.1 ODCIIndexStart()

`ODCIIndexStart()` iniciuje vyhodnocování operátoru (v našem případě `s_point_contain()` – viz *kap. 5.6.4*), jehož první parametr je sloupec, který má nad sebou definovaný doménový index souvisejícího indexového typu.

Konkrétní parametry této metody jsou následující:

- `sctx (IN OUT)` typu `DomIndex` (vytvořený implementační typ) – využívaný je pouze `OUT`, kterým se dalším metodám pro prohledávání indexu předávají stavové informace o indexu a operátorech.
- `ia`, `env` – mají stejnou funkci jako u ostatních metod.
- `op` typu `ODCIPredInfo` – obsahuje metadata souvisejícího operátoru (zde konkrétně `s_point_contain`).
- `qi` typu `ODCIQueryInfo` – obsahuje dodatečné informace o dotazu.
- `strt`, `stop` – jsou parametry stejného typu jako návratový typ operátoru, v tomto případě `NUMBER`, které označují horní a dolní hranici operátorem vracených hodnot.
- Na tomto místě musí být specifikovány všechny další (kromě prvního – indexovaného sloupce) parametry vyhodnocovaného operátoru. Konkrétně jde o dva argumenty `cmpval` typu `pointsS` a `cmpvalT` typu `pointsT` (viz *kap. 5.6*).

Tato metoda se může v doménovém indexu nacházet vícekrát, každá pro použití s jedním konkrétním operátorem. V tomto doménovém indexu je však použita pouze jedna, starající se o prohledávání indexu při použití popsaného operátoru.

`ODCIIndexStart()` funguje ve spojitosti s operátorem `s_point_contain()`, který se dotazuje na čísla všech trajektorií, které prošly daným místem na mapě v určitém čase. Tato dotazovaná hodnota je předávána pomocí zmíněných parametrů `cmpval` a `cmpvalT`. Nejdříve se tedy spustí prohledávání indexu. Je otevřen kurzor pro tabulku s uzly a nalezeny všechny záznamy, jejichž listy se v daném časoprostorovém okně nachází. Poté je otevřen kurzor i pro tabulku listů a z určených listů jsou vybrána ROWID řádků v základní tabulce a kontext je parametrem `sctx` předán metodě `ODCIIndexFetch()`.

5.5.3.2 ODCIIndexFetch()

Tato metoda je používána v úzkém spojení s metodou `ODCIIndexStart()`.

Oproti `ODCIIndexStart()` využívá tyto nové parametry:

- `SELF` (`IN OUT`) typu `DomIndex` – kontext vyhledávání odesílán (`OUT`) dalšímu volání této metody (`IN`).
- `nrows` – značí maximální počet výsledných řádků, který může být jedním voláním této metody vrácen.

- `rids` (`OUT`) typu `ODCIRidList` – obsahuje kolekci ROWID vracených řádků základní tabulky.

`ODCIIndexFetch()` při každém svém volání vrátí výslednou kolekci `rids` v maximálním počtu `nrows`., jako odpověď pro volaný operátor (`s_point_contain()`). To se opakuje tak dlouho, dokud nejsou vráceny všechny odpovídající řádky, tedy do vrácení hodnoty `NULL`.

5.5.3.3 ODCIIndexClose()

Tato metoda ukončí vyhodnocování operátoru uzavřením kurzoru `cnum`.

5.6 Operátory

Samotné operátory, sloužící pro dotazování nad indexovanou tabulkou s využitím doménového indexu, jsou použity čtyři. Všechny tyto operátory jsou přiřazeny k indexovému typu `IType` a každý operátor je také svázán s funkcí, která jej implementuje.

Jako hlavní typy, se kterými operátory pracují, jsou zavedeny `pointsS`, který reprezentuje prostorový rozsah a `pointsT` jako časový interval. K vracení záznamů trajektorie s požadovanými segmenty je použitý typ `trajectory_entries_nt`, který obsahuje kolekci záznamů s číslem trajektorie a jejich nalezených segmentů.

První tři operátory `r_query()`, `top_query()` a `combined_query()` jsou použity mimo klauzuli `WHERE`, jsou tedy vyhodnocovány na základě jejich funkční implementace. Poslední operátor, použitý s klauzulí `WHERE`, je vyhodnocován jak jeho funkční implementací, tak i metodami pro prohledávání indexu.

Příklady konkrétních dotazů jsou uvedeny v *kap. 6*.

5.6.1 r_query()

```
CREATE OPERATOR r_query
BINDING (pointsT, pointsS) RETURN trajectory_entries_nt
WITH INDEX CONTEXT, SCAN CONTEXT DomIndex
USING pt_inside;
```

Kód 5.7: Vytvoření operátoru r_query

Tento operátor vykonává rozsahový dotaz. Zadáním dvou argumentů typů `pointsT` a `pointsS` specifikujeme časoprostorové okno. S operátorem svázaná funkce `pt_inside()` pak prochází TB-strom, vyhledá všechny segmenty nacházející se v zadaném okně a vrátí je v kolekci spolu s čísly jejich trajektorií.

5.6.2 top_query()

```
CREATE OPERATOR top_query
BINDING (pointsT, pointsS, VARCHAR2) RETURN ntab
WITH INDEX CONTEXT, SCAN CONTEXT DomIndex
USING top_query_f;
```

Kód 5.8: Vytvoření operátoru top_query

Operátor `top_query()` zastupuje topologický typ dotazu. Jako argumenty opět přijímá časoprostorové okno vyjádřené typy `pointsT` a `pointsS` a navíc příznak, který může nabývat (podle *kap. 4.2.*1) jednu ze tří hodnot: `E`, `L` nebo `EL`.

- `E` – hledáme trajektorie, které do daného časoprostorového okna vstupují
- `L` – hledáme trajektorie, které dané časoprostorové okno opouští
- `EL` – hledáme trajektorie, které daným časoprostorovým oknem prochází, tedy do něj vstupují i z něj vystupují.

Operátor vrátí čísla všech trajektorií, které splňují dané podmínky.

5.6.3 combined_query()

```
CREATE OPERATOR combined_query
BINDING (pointsT, pointsS, pointsT, pointsS) RETURN
  trajectory_entries_nt
USING combined_query_f;
```

Kód 5.9: Vytvoření operátoru combined_query

Pro vykonávání kombinovaného typu dotazu slouží operátor `combined_query()`. Jak vyplývá z *kap. 4.3.3.2*, tento operátor přijímá jako argumenty ne jedno, ale dvě časoprostorová okna. V prvním kroku jsou záznamy omezeny na ty, které splňují první podmínku, tedy v našem případě náležitost k prvnímu oknu. V druhém kroku se pak z těchto segmentů za pomoci zřetězeného seznamu v listech stromu vyberou ty, které odpovídají druhé podmínce, tedy druhému oknu. Tímto způsobem jsou vráceny všechny odpovídající části trajektorií.

5.6.4 s_point_contain()

```
CREATE OPERATOR s_point_contain
BINDING (trajectory_entries, pointsS, pointsT) RETURN
  NUMBER
USING s_point_contain_f;
```

Kód 5.10: Vytvoření operátoru s_point_contain

I zde se jedná o rozsahový typ dotazu, který se ptá na čísla všech trajektorií, které se vyskytovaly v daném časoprostorovém okně. Tento operátor se ovšem používá v klauzuli `WHERE` a jeho první parametr je typu sloupce, nad kterým je vybudován doménový index. Splňuje tak podmínky nutné k tomu, aby mohl být vyhodnocen metodami pro prohledávání indexu. Na druhé straně pro něj ale existuje i funkční implementace v podobě svázané funkce `s_point_conatin_f()`. Rozhodnutí, který způsob vyhodnocení této metody bude použit, provede optimalizátor na základě jeho selektivních a nákladových funkcí.

5.7 Funkční implementace

Pro vyhodnocování operátorů je ale nejdříve nutné implementovat funkce, které budou požadované operace vykonávat. Část kódu, která takto podporuje operátory, se tedy nazývá funkční implementace. V následující části budou tyto konkrétní funkce svázané s výše prezentovanými operátory popsány.

5.7.1 pt_inside()

Tato funkce implementuje rozsahové vyhledávání v TB-stromu. Prochází všechny uzly stromu, a pokud je nalezen uzel, který se překrývá (funkce `Overlap()`) s hledaným časoprostorovým oknem, rekurzivně se projdou všechny jeho potomci. Pokud se dostaneme na uzel, jehož hloubka je ve stromu rovna 1, znamená to, že uzel již ukazuje na list. Z tohoto listu se vyberou jednotlivé segmenty překrývající se s dotazovaným oknem a přidají se do sady výsledků. Celý tento rekurzivní proces na konci vrátí operátoru kolekci segmentů s příslušností k jejich trajektorii výše popsaného typu `trajectory_entries_nt`.

5.7.2 top_query_f()

Implementace této funkce vychází z *kap. 4.3.3.1*, kde byl popsán princip topologických dotazů. `Top_query_f()` opět podle překrývání s dotazovaným oknem rekurzivně prochází strom až k listům, u kterých se testuje průnik jejich jednotlivých záznamů. Pokud je první záznam v prvním nalezeném listu obsažen v požadovaném okně, znamená to, že trajektorie může okno už jen procházet nebo jej opouštět. Rozlišení mezi těmito dvěma stavy provedeme až testováním posledního záznamu v posledním nalezeném listu dané trajektorie. V případě, že se i tento záznam nachází v hledaném okně, znamená to, že trajektorie toto okno prochází (`EL`), v opačném případě trajektorie toto okno opouští (`L`). Jestliže se první záznam v prvním nalezeném listu nenachází v daném okně, pak to může znamenat, že trajektorie do okna vstupuje, nebo se v něm vůbec nevyskytuje. I zde testujeme průnik posledního záznamu posledního nalezeného listu dané trajektorie. Pokud je v okně obsažen, pak

trajektorie do okna vstupuje (`E`), jinak trajektorie okno obchází, tedy s ním nemá v překrytí žádný svůj segment.

Podle těchto podmínek je v souvislosti se zadaným příznakem (`E`, `L` nebo `EL`) vrácena kolekce identifikátorů všech vyhovujících trajektorií.

5.7.3 combined_query_f()

Jedná se o funkci implementující algoritmus kombinovaného vyhledávání (*Kód 4.3*).

Funkce nejdříve stejným způsobem jako `pt_inside()` rekurzivně prochází celým stromem a vyhledává segmenty odpovídající prvnímu časoprostorovému oknu. Poté se zjistí první a poslední bod spojených segmentů a od těchto bodů se postupuje dále pomocí zřetězeného seznamu.

Nejdříve je následován poslední bod, od kterého se prochází směrem dopředu až po poslední záznam daného listu. Z posledního záznamu se pak na další list přejde ukazatelem `ptrNext`. Trajektorie se tak prochází do té doby, až je nalezen její poslední segment překrývající se se zadaným oknem. Celý výsledek je pak zaznamenán do kolekce záznamů typu `trajectory_entries_nt`.

Ve druhé fázi se pak obdobným způsobem postupuje od počátečního bodu nalezených segmentů směrem zpět podle ukazatele `ptrPrevious` až po první (nejstarší) vyhovující segment. Výsledek je pak přidán do kolekce záznamů typu `trajectory_entries_nt` a vrácen operátoru.

5.7.4 s_point_contain_f()

Funkce `s_point_contain_f()` je funkční implementací operátoru `s_point_contain()`. Tento operátor je však vyhodnocován metodami pro prohledávání indexu, tedy tato implementace je při použití doménového indexu redundantní. V případě, že však není doménový index dostupný, pak je tato funkce využita a vrací operátoru trajektorie přímo ze základní tabulky bez použití indexu.

5.8 Indexový typ

Konkrétní indexový typ doménového indexu používaný při vytváření indexu je vytvořen takto:

```
CREATE INDEXTYPE IType FOR
  r_query(pointsT, pointsS),
  top_query(pointsT, pointsS, VARCHAR2),
  combined_query(pointsT, pointsS, pointsT, pointsS),
  s_point_contain(trajectory_entries, pointsS, pointsT)
USING DomIndex
WITH SYSTEM MANAGED STORAGE TABLES;
```

Kód 5.11: Vytváření indexového typu

V tomto indexovém typu jsou specifikovány všechny podporované operátory a jeho implementační typ `DomIndex`. Poslední řádek udává, že doménový index je řízený systémem, využívá tedy doporučovaný přístup, který byl uvedený ve verzi Oracle 11g release 1.

6 Použití a testování doménového indexu

Pro použití indexu je tedy vytvořena základní tabulka trajektorií `trajTable`, která je následně naplněna daty ve tvaru: id objektu, id trajektorie, datum, čas, zeměpisná šířka, zeměpisná délka:

```
0862;1;10/09/2002;09:15:59;23.845089;38.018470;
0862;1;10/09/2002;09:16:29;23.845179;38.018069;
0862;1;10/09/2002;09:16:59;23.845530;38.018241;
...
```

Tento soubor dat tvoří celkem 112 203 záznamů. Interval mezi jednotlivými časovými razítky je 30 s. Celkem obsahuje 276 trajektorií s průměrným počtem 407 záznamů.

Doménový index pak může být vytvořen příkazem:

```
CREATE INDEX idx1 ON trajTable(trajectory)
INDEXTYPE IS userT.IType;
```

Kód 6.1: Vytvoření doménového indexu

Doménový index lze samozřejmě také vytvořit nad prázdnou tabulkou a následně poté ji začít naplňovat daty nebo ji kdykoliv po vytvoření indexu ještě doplnit. Pro takový případ jsou implementovány metody `ODCIIndexInsert()` a `ODCIIndexUpdate()`.

6.1 Použitý hardware

Veškeré práce byly prováděny s využitím databázového serveru Oracle 11g release 2, který běžel na notebooku s operačním systémem Windows 7 Professional SP1 s následující konfigurací:

Procesor: Intel Pentium M, 1,6 GHz

HDD: Hitachi, 60 GB, 5400 rpm

RAM: 2 GB, DDR 3

6.2 Vypočítání statistik doménového indexu

Výpočet statistik doménového indexu se provádí voláním procedury `gather_table_stats()` z balíčku `DBMS_STATS` nad základní indexovanou tabulkou `trajTable`.

Toto volání využívá typ `my_stats`, který implementuje tři funkce – `ODCIGetInterface()`, `ODCIStatsCollect()` a `ODCIStatsDelete()`, které jednoduše

vypočítají základní statistiky pro jednotlivé indexové tabulky (`ODCIStatsCollect()`), případně vytvořené statistiky při smazání indexu odstraní (`ODCIStatsDelete()`). Nakonec se provede přiřazení statistik k danému indexovému typu:

```
ASSOCIATE STATISTICS WITH INDEXTYPES IType USING my_stats
WITH SYSTEM MANAGED STORAGE TABLES;
```

6.2: Přiřazení statistik indexovému typu

Rozsáhlé možnosti optimalizátoru pro rozšířené indexování jsou rozebírány v dokumentaci Oracle [16].

6.3 Velikost indexu

Velikost indexu se samozřejmě odvíjí od počtu záznamů a je dána velikostí všech indexových tabulek. Příklady konkrétních hodnot jsou uvedeny v *Tab. 6.1*. Tyto hodnoty byly zjištěny z pohledů na `dba_all_tables` po vypočítání příslušných statistik.

Počet záznamů v základní tabulce	Počet nelistových uzlů	Počet listových uzlů	Celková velikost indexu (kB)
10 000	5	189	53
50 000	19	952	243
100 000	36	1915	459

Tab. 6.1: Velikost indexu

Z tabulky je vidět, že velikost indexu v poměru k jednomu záznamu klesá. To je z toho důvodu, že se zvyšujícím se počtem záznamů se stále efektivněji zaplňuje indexová struktura. Procento zaplnění se limitně blíží k 100 %.

6.4 Rychlost vkládání

Rychlost vkládání různých počtů záznamů je opět pro lepší představu zobrazena v *Tab. 6.2*:

Počet vkládaných záznamů	Průměrná doba vkládání (s)
10 000	9
20 000	19
30 000	30

Tab. 6.2: Rychlost vkládání

Z těchto hodnot je tedy vyvozeno, že průměrná rychlost vkládání je asi 1 000 záznamů za sekundu. Vzhledem k tomu, že stávající databáze je tvořena záznamy 50 nákladních vozidel, které jsou přidávané vždy jedenkrát za 30 sekund, je tato rychlost více než dostačující.

Testování struktury TB-stromu však není cílem této práce (testy TB-stromu byly provedeny spolu s jeho uvedením v [10]. Zaměřme se teď proto na dotazování nad implementovaným doménovým indexem.

6.5 Vyhodnocování dotazů

Základní úkol doménových indexů je zjednodušit a zefektivnit práci, tedy v první řadě dotazování, s komplexními nebo nějakým způsobem specifickými daty. Proto by mělo být testováno především správné a rychlé vyhodnocování uživatelských dotazů nad indexovanou tabulkou.

V této kapitole bude pro každý druh dotazu uveden konkrétní příklad tak, aby byl využit každý implementovaný operátor. Dotaz bude popsán slovně i jazykem SQL a bude také popsán způsob jeho vyhodnocení v souvislosti s doménovým indexem. Pro zjištění některých podrobností je využíván příkaz `EXPLAIN PLAN` a následně tabulka `plan_table`.

Kompletní odpovědi na jednotlivé dotazy lze nalézt v příloze.

6.5.1 Rozsahový dotaz

„Kudy přesně v daném území projížděla všechna nákladní vozidla, která se zde pohybovala dnes od 12:30 do 13 hodin?“

```
SELECT r_query(
  pointsT('10/09/2002 13:40:00','10/09/2002 14:40:00'),
  pointsS(23.800122,38.018069,23.860539,38.191064) )
FROM DUAL;
```

Kód 6.3: Rozsahový dotaz

Způsob vyhodnocení: funkční implementací využívající doménový index.

Tento operátor můžeme využít i k tzv. ***time-slice*** dotazu, kdy se netážeme na časový úsek, ale na časový bod. Takový dotaz bude mít oba časové body shodné. Např.: „*Které nákladní vozy se na daném místě vyskytují právě teď*?“

6.5.1.1 Rozsahový dotaz pomocí metod pro prohledávání indexu

Druhým rozsahovým dotazem by mohl být například:

„*Vyber všechny trajektorie nákladních vozidel, která navštívila určité místo v daném čase*.“

```
SELECT trID FROM trajtable WHERE
  s_point_contain(
    trajectory,
    pointsS(23.855550,38.110650,23.875550,38.930650),
    pointsT('10.09.2002 13:00:00','17.09.2002 13:00:00')
  )=1
ORDER BY trID;
```

Kód 6.4: Rozsahový dotaz využívající metody pro prohledávání indexu

Způsob vyhodnocení: metodami pro prohledávání indexu.

6.5.1.2 Rozsahový dotaz bez použití indexu

Pokud index není vytvořen nebo je smazán nebo v případě, pak je k vyhodnocení předcházejícího dotazu použita místo metod pro prohledávání indexu jeho funkční implementace (funkce `s_point_contain_f()`). Ta se v tomto případě dotazuje přímo na základní tabulku bez použití doménového indexu.

V případě, že index nad tabulkou existuje, rozhodne o způsobu jeho vyhodnocení vestavěný optimalizátor, případně implementovaný optimalizátor pro rozšířené indexování.

Jednoduchý výpis z tabulky `plan_table`, která mimo jiné ukazuje, jaký index byl při dotazu použit, je dostupný v příloze.

6.5.2 Topologický dotaz

„Najdi trajektorie nákladních vozidel, která opustila dané místo mezi 7 a 9 hodinou ranní dne 13. 9. 2002.“

```
SELECT top_query(
  pointsT('13/09/2002 07:00:00','13/09/2002 09:00:00'),
  pointsS(23.531865,38.021547,23.831865,38.119571),
  'L')
FROM DUAL;
```

Kód 6.5: Topologický dotaz

Způsob vyhodnocení: funkční implementací využívající doménový index.

6.5.3 Kombinovaný dotaz

„Kudy v určité oblasti dnes od 8 do 16 hodin projížděly nákladní automobily, které byly dnes od 10 do 11 hodin na daném místě?“

```
SELECT combined_query(
```

```
  pointsT('11/09/2002 10:00:00','11/09/2002 11:00:00'),
  pointsS(23.079288,38.029928,23.916981,38.116981),
  pointsT('11/09/2002 08:00:00','11/09/2002 16:00:00'),
  pointsS(23.795179,38.044780,23.918069,38.014018))
FROM DUAL;
```

Kód 6.6: Kombinovaný dotaz

Způsob vyhodnocení: funkční implementací využívající doménový index.

Druhé omezení nemusí představovat jen časoprostorové okno. Objekt lze vybrat i použitím některého jiného operátoru. Takto, spojením více operátorů, mohou být vytvořeny další kombinované dotazy.

6.5.4 Členské metody indexovaného sloupce

K dodatečným operacím s trajektoriemi mohou být navíc využity následující členské metody implementované typem `trajectory_entries`.

- **`Get_traj`** – pouze vrátí všechny záznamy zadané trajektorie.
- **`Get_avg_speed`** – vypočítá průměrnou rychlost v dané trajektorii.
- **`Get_distance`** – určí délku celé trajektorie, tedy vzdálenost ujetou objektem.

6.6 Smazání indexu

Nakonec může být příkazem `DROP INDEX idx1` provedeno smazání indexu. Tím jsou spuštěny metody `ODCIIndexDelete()` a `ODCIStatsDelete()`, které vymažou indexové tabulky a přiřazené statistiky.

6.7 Rychlost dotazování

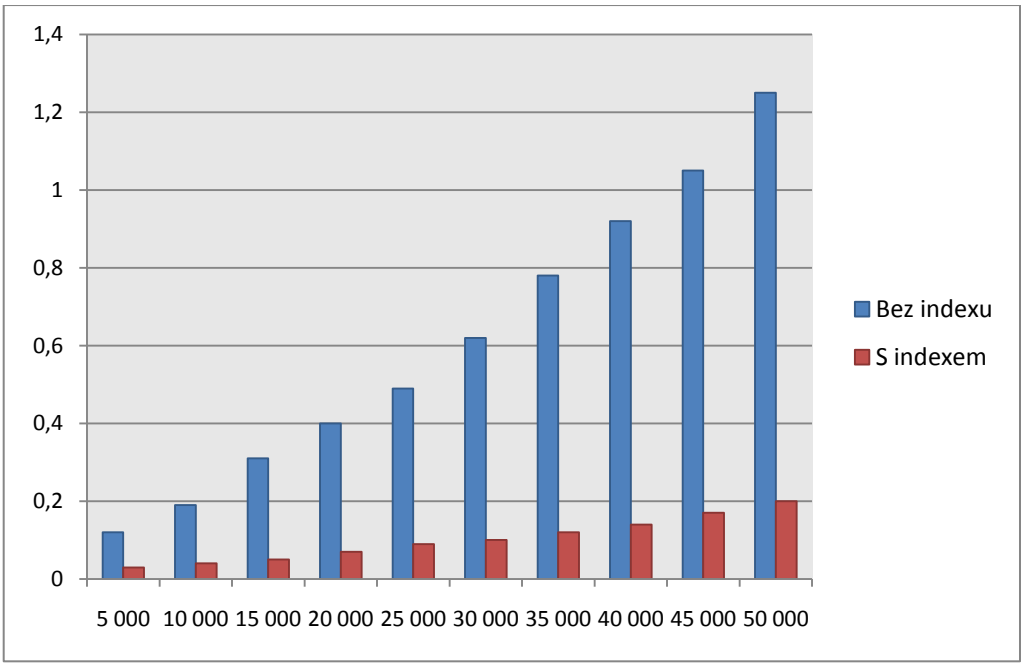
Abychom ověřili, zda doménový index skutečně pomůže urychlit konkrétní dotazy nad trajektoriemi, je třeba porovnat rychlost konkrétního dotazu bez použití a následně s použitím indexu. Toto je prováděno s využitím rozsahového dotazu. Průměrné naměřené hodnoty jsou zaznamenány v tabulce (*Tab. 6.3*).

Počet záznamů	Bez použití indexu (s)	S použitím indexu (s)
5 000	0,12	0,03
10 000	0,22	0,04
15 000	0,33	0,05
20 000	0,45	0,07

25 000	0,57	0,09
30 000	0,79	0,10
35 000	1,09	0,12
40 000	1,55	0,14
45 000	1,78	0,17
50 000	2,07	0,20

Tab. 6.3: Porovnání rychlosti dotazů

Naměřené hodnotu jsou pro porovnání znázorněny v následujícím grafu (*Graf 1*).



Graf 1: Porovnání rychlosti dotazů

Jak je z grafu patrné, vytvořený doménový index skutečně značně pozitivně ovlivňuje rychlost dotazování. Horizontální osa značí počet záznamů v tabulce a vertikální osa potřebný čas v sekundách. Při malém počtu záznamů ještě rozdíl není tak patrný a pravděpodobně bychom se obešli i bez indexu, ale s rostoucím objemem dotazovaných dat se poměr rychlosti dotazů nad neindexovanými daty k rychlosti dotazů nad daty s indexem dramaticky zvyšuje. Můžeme vidět, že například při 50 000 záznamech přináší doménový index zhruba šestinásobné urychlení dotazu.

Při diskuzi výsledků je třeba brát v úvahu, že se jedná pouze o průměrné hodnoty. Jako takové mohou být ovlivněny parametry dotazu, tedy v tomto případě především velikostí časoprostorového okna. Praxe ukazuje, že se zvětšujícím se rozsahem se rozdíly v rychlosti vyrovnávají. Dalším výrazným činitelem, který může mít na výsledky měření na osobním počítači zásadní vliv, je množství nesouvisejících procesů běžících na pozadí. I přes tyto ovlivňující faktory však zůstává tendence grafu, ukazující urychlení dotazů při použití doménového indexu, zachována.

7 Návrhy pro rozšíření

Vzhledem ke značným možnostem rozšířeného indexování v databázi Oracle se nabízí celá řada rozšíření, případně změn pro navržený doménový index.

7.1 Vybudování databáze

Doménové indexy si kladou za cíl urychlit a celkově zefektivnit práci se specializovanými databázemi. To znamená, že primárně by měla být vyvinuta databáze sloužící k určitému účelu, a teprve až poté by měla být podpořena doménovým indexem, který by byl připraven přesně podle požadavků této databáze. Implementovaný doménový index může být teoreticky využit v praxi, ale bude k němu nejdříve třeba nalézt, případně naimplementovat, databázi specifické domény, která by jeho možnosti dokázala efektivně využívat. V takovém případě by bylo nutné databázi implementovat s ohledem na stávající doménový index nebo upravit funkčnost doménového indexu tak, aby vyhovoval nové databázi.

7.2 Širší podpora pohybujících se objektů

Implementovaná struktura TB-stromu indexuje pohybující se objekty, které chápe pouze jako bezrozměrné body v prostoru. Tento postup je pro konkrétní případ dostačující, ovšem pro databázi, která by potřebovala zpracovat i objekty, u kterých by byly důležité jejich konkrétní rozměry, by byl tento doménový index nedostačující. Proto je další možností pro rozšíření realizovat podporu objektům o určitých rozměrech a tvarech, čímž by bylo následně umožněno rozšířit funkčnost podporované databáze.

7.3 Lepší využití optimalizátoru

Další možností pro zaměření budoucího vývoje by mohlo být větší využití optimalizátoru pro rozšířené indexování. Možnosti tohoto optimalizátoru jsou značně rozsáhlé. Pro větší zefektivnění dotazování by se tedy dalším krokem mohla stát implementace specializovaných cenových a selektivních funkcí.

7.4 Aplikační nadstavba

Aplikace pro samotný doménový index by neměla smysl. Ovšem s vytvořením konkrétní specializované databáze, která by například měla sloužit manažerovi kurýrní společnosti pro

zaznamenávání a plánování tras rozvážek, by mohla být těžištěm funkčnosti vizualizace výsledků na displeji. Uživatel by pak mohl snadno zjišťovat kudy, v jakém čase, případně jak rychle se konkrétní zaměstnanci pohybovali. Z tohoto pohledu by byla aplikační nadstavba pro doménový index, resp. databázi, nezbytná.

7.5 Jiná struktura

S ohledem na rychlý vývoj v oblasti indexování časoprostorových dat, kdy se neustále objevují nové efektivnější struktury pro práci s těmito daty, je třeba předpokládat příchod struktury, která bude pro konkrétní účely vhodnější než implementovaný TB-strom. Už nyní existují mnohé zajímavé struktury, které je také možné pro indexování časoprostorových dat účelně využít. Mimo jiné například MV3R-tree, H-R-tree, SEB-tree, BB^x-index a další (viz [9] nebo [14]). Kteroukoliv z těchto struktur je samozřejmě možné využít pro vybudování doménového indexu.

8 Závěr

Cílem této diplomové práce bylo nejdříve se seznámit se způsoby indexování dostupnými u databázového serveru Oracle 11g a následně podrobně popsat způsob tvorby doménových indexů. Dále pak vybrat typ indexovací struktury a tuto strukturu v podobě doménového indexu implementovat a vhodným způsobem otestovat.

Nejdříve bylo dohodnuto, že doménový index bude pracovat s časoprostorovými daty. Pro taková data se nabízí celá řada vhodných stromů, z nichž byl následně vybrán TB-strom.

Dále bylo nutné zvolit zdrojová data, která budou doménovým indexem zpracovávána. Vhodná data byla nalezena v [22]. Jedná se o reálné záznamy trajektorií nákladních vozidel. Po vytvoření databáze a její naplnění těmito daty bylo možné začít vytvářet samotný doménový index.

Vzhledem k tomu, že kromě dokumentace Oracle nebyly dostupné jiné funkční implementace nebo příklady doménového indexu, bylo počáteční seznamování s rozhraním ODCI velmi zdlouhavé.

Prvním krokem implementace bylo každý objekt, se kterým bude pracováno, reprezentovat jeho vlastním objektovým typem. Především se jednalo o vytvoření objektových typů pro pohybující se objekty a jednotlivé části struktury TB-stromu. Klíčová metoda doménového indexu je `ODCIIndexInsert()`, která se stará o vytvoření indexových tabulek a jejich naplnění daty. Pro tuto metodu bylo tedy třeba implementovat algoritmy vykonávající vkládání do TB-stromu, aby byly indexové tabulky vhodným způsobem naplněny. Tyto algoritmy jsou následně využívány i dalšími metodami podílejícími se na údržbě indexu.

Aby bylo možné otestovat, že takto vytvořený index funguje, bylo třeba navrhnout různé dotazy nad indexovanými daty. Proto bylo vytvořeno několik operátorů, svázaných funkcí a metod ODCI, které tyto dotazy reprezentují. Následné prohledání indexu a zodpovězení dotazu může učinit buď konkrétní funkce svázaná s operátorem, nebo metody z rozhraní ODCI.

V této práci byla tedy konkrétně popsána a realizována implementace doménového indexu, která ukazuje a vysvětluje na konkrétních případech způsob využití rozhraní pro rozšířené indexování. Jako taková může v budoucnu sloužit nejen pro urychlení přístupu do databáze, ale sekundárně i jako jeden z mála konkrétních příkladů implementace doménových indexů a výrazně tak pomoci příštím programátorům vlastních doménových indexů.

Literatura

[1] Bryla, B., Loney, K. *Mistrovství v Oracle Database 11g*. 1. vydání. Brno : Computer Press, 2009. ISBN 978-80-251-2189-4.

[2] Cha, C., Kim, S., Won, J., Lee, J., Bae, D. Efficient Indexing in Trajectory Databases. *International Journal of Database Theory and Application* [online]. 2005, vol. 1, no. 1, [cit. 2010-11-18]. Dostupné z: *http://www.sersc.org/journals/IJDTA/vol1_no1/papers/03.pdf*

[3] Egenhofer, M., Franzosa, R. Point-Set Topological Spatial Relations. *International Journal of Geographical Information Science*, 1991, vol. 5, no. 2, s. 161-172.

[4] Erwig, M., Schneider, M. *Developments in Spatio-Temporal Query Languages.* In *Proceedings of DEXA Workshop on Spatio-Temporal Data Models and Languages*, 1999.

[5] Frentzos, K. *Trajectory Data Management in Moving Object Databases : Ph.D. Thesis.* Athens : University of Piareus, Department of Informatics, 2008.

[6] Gianotti, F. *Mobility, Data Mining and Privacy.* Berlin : Springer-Verlag, 2008. ISBN 978-3-540-75176-2.

[7] Greenwald, R., Stackowiak, R., Stern J. *Oracle Essentials: Oracle Database 11G*. 4th Edition. Sebastopol : O'Reilly Media, 2008. ISBN 978-0-596-51454-9.

[8] Guttman, A. *R-trees: A Dynamic Index Structure for Spatial Indexing*. Berkeley : University of California, 1984.

[9] Illari, S., Mena, E., Illarramendi, A. Location-Depend Query Processing: Where We Are and Where We Are Heading. *ACM Computing Surveys*, March 2010, vol. 42, issue 3.

[10] Jennsen, Ch., Pfoser, D., Theodoridis, Y. *Novel Approaches to the Indexing of Moving Object Trajectories.* In *Proceedings of the 26th International Conference on Very Large Databases*, Cairo, 2000.

[11] Kaster, D., Bugatti, P., Traina, A. FMI-SiR: A Flexible and Efficient Module for Similarity Searching. *Journal of Information and Data Management*, 2010, vol. 1, no. 2.

[12] Loney, K. *Oracle Database 11g: The Complete Reference*. 1st Edition. New York : McGraw-Hill, 2009. ISBN 978-0-07-159876-7.

[13] Marcos, E., Vela, B., Cavero, J., Cáceres, P. *Aggreggation and Composition in Object-Relational Database Design* [online]. 2000 [cit. 2011-02-10]. Dostupné z:

http://www.downloadic.com/miia1lt/Aggregation-and-composition-in-object-relational-database-design.html

[14] Mokbel, M., Ghanem, T., Aref. W. *Spatio-Temporal Access Methods.* In *IEEE Computer Society Technical Committee on Data Engineering*, 2003.

[15] Oracle. *Oracle® Database Concepts* [online]. 2010 [cit. 2010-06-22]. Dostupné z: http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/toc.htm.

[16] Oracle. *Oracle® Database Data Cartridge Developer's Guide* [online]. 2011 [cit. 2011-03-12]. Dostupné z: http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10765/toc.htm.

[17] Oracle. *Oracle® Database Object-Relational Developer's Guide* [online]. 2011 [cit. 2011-01-10]. Dostupné z: http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11822/adobjxmp.htm.

[18] Oracle. *Oracle® Database PL/SQL Language Reference* [online]. 2011 [cit. 2011-01-30]. Dostupné z: http://download.oracle.com/docs/cd/E11882_01/appdev.112/e17126/toc.htm.

[19] Oracle. *Oracle® Database SQL Language Reference* [online]. 2011 [cit. 2010-12-29]. Dostupné z: http://download.oracle.com/docs/cd/E11882_01/server.112/e17118/toc.htm.

[20] Pelekis, N., Frentzos, E., Theodoridis, Y., Giatrakos, N. *Supporting Movement in ORDBMS – the HERMES MOD Engine*. Hellas: University of Piraeus, Department of Informatics, 2006.

[21] Pfoser, D. *Issues in the Management of Moving Points : Ph.D. Thesis*. Aalborg : Aalborg University, Computer Science Department, 2002.

[22] Theodoridis, Y. *Spatio-temporal (trajectory) datasets* [online]. 2006 [cit. 2010-11-18]. Dostupné z: http://www.rtreeportal.org/index.php?option=com_content&task=view&id=30&Itemid=43.

Seznam příloh

Příloha A: Příklady SQL dotazů

Rozsahový dotaz:

```
SELECT r_query(
  pointsT('10.09.2002 13:40:00','10.09.2002 14:40:00'),
  pointsS(23.800122,38.018069,23.860539,38.191064)
)
FROM DUAL;

--> záznamy s příslušností k trajektorii
TRAJECTORY_ENTRIEST(TRAJECTORY_ENTRYT(
POINTSST(
POINTST('10.09.02 14:05:44', '10.09.02 14:06:14'),
POINTSS('23,795559', '38,096194', '23,8018', '38,098774')),
POINTSST(
POINTST('10.09.02 14:06:14', '10.09.02 14:06:44'),
POINTSS('23,8018', '38,098774', '23,8079', '38,101192')),
...
POINTSST(
POINTST('10.09.02 14:39:14', '10.09.02 14:39:44'),
POINTSS('23,840635', '38,131751', '23,839636', '38,131522')),
POINTSST(
POINTST('10.09.02 14:39:44', '10.09.02 14:40:14'),
POINTSS('23,839636', '38,131522', '23,837986', '38,128181'))),
4),
TRAJECTORY_ENTRIEST(TRAJECTORY_ENTRYT(
POINTSST(
POINTST('10.09.02 13:42:52', '10.09.02 13:43:22'),
POINTSS('23,851974', '38,017661', '23,850855', '38,020191')),
...
POINTSST(
POINTST('10.09.02 14:12:22', '10.09.02 14:12:52'),
POINTSS('23,850667', '38,02061', '23,852497', '38,016571'))),
10),
TRAJECTORY_ENTRIEST(TRAJECTORY_ENTRYT(
POINTSST(
POINTST('10.09.02 14:02:43', '10.09.02 14:03:13'),
POINTSS('23,861147', '38,020038', '23,860348', '38,020248')),
...
POINTSST(
POINTST('10.09.02 14:31:43', '10.09.02 14:32:13'),
POINTSS('23,859858', '38,020214', '23,860878', '38,019992'))),
12)
```

Rozsahový dotaz s operátorem v klauzuli `WHERE`

```
SELECT trID FROM trajtable WHERE
  s_point_contain(trajectory,pointsS(23.855550,38.110650,23.875550,
  38.930650),pointsT('10.09.2002 13:00:00','17.09.2002 13:00:00'))=1
ORDER BY trID;

--> čísla trajektorií
4, 5, 8, 9
```

```
--> následný výpis z PLAN_TABLE:
OPERATIONS         OPTIONS     OBJECT_NAME
------------------ ----------- -----------
SELECT STATEMENT
SORT               ORDER BY
TABLE ACCESS       BY ROWID    TRAJTABLE
DOMAIN INDEX                   IDX1
-----------------------------------------
```

Rozsahový dotaz s operátorem v klauzuli `WHERE` bez použití indexu

```
-- smažeme index a položíme stejný dotaz jako výše:
SELECT trID FROM trajtable WHERE
  s_point_contain(trajectory,pointsS(23.855550,38.110650,23.875550,
  38.930650),pointsT('10.09.2002 13:00:00','17.09.2002 13:00:00'))=1
ORDER BY trID;

--> čísla trajektorií
4, 5, 8, 9


--> následný výpis z PLAN_TABLE:
OPERATIONS         OPTIONS     OBJECT_NAME
------------------ ----------- -----------
SELECT STATEMENT
SORT               ORDER BY
TABLE ACCESS       FULL        TRAJTABLE
-----------------------------------------
```

Time-slice dotaz:

```
SELECT r_query(
  pointsT('12.09.2002 15:00:00','12.09.2002 15:00:00'),
  pointsS(23.064251,38.123959,23.960539,38.791064))
FROM DUAL;

--> záznamy s příslušností k trajektorii
TRAJECTORY_ENTRIEST(TRAJECTORY_ENTRYT(
POINTSST(
  POINTST('12.09.02 14:59:46', '12.09.02 15:00:16'),
  POINTSS('23,840911', '38,131278', '23,840542', '38,131789'))),
6)
```

Topologický dotaz:

```
SELECT top_query(
  pointsT('13/09/2002 07:00:00','13/09/2002 09:00:00'),
  pointsS(23.531865,38.021547,23.831865,38.119571),
  'L')
FROM DUAL;

--> čísla trajektorií
7
```

Kombinovaný dotaz:

```
SELECT combined_query(
  pointsT('11/09/2002 10:00:00','11/09/2002 11:00:00'),
  pointsS(23.079288,38.029928,23.916981,38.116981),
  pointsT('11/09/2002 08:00:00','11/09/2002 16:00:00'),
  pointsS(23.795179,38.044780,23.918069,38.014018))
FROM DUAL;

--> záznamy s příslušností k trajektorii
TRAJECTORY_ENTRIEST(TRAJECTORY_ENTRYT(
POINTSST(
  POINTST('11.09.02 08:03:16', '11.09.02 08:03:46'),
  POINTSS('23,837914', '38,045654', '23,836532', '38,044723')),
POINTSST(
  POINTST('11.09.02 08:03:46', '11.09.02 08:04:16'),
  POINTSS('23,836532', '38,044723', '23,835932', '38,045123')),
...
POINTSST(
  POINTST('11.09.02 08:34:16', '11.09.02 08:34:46'),
  POINTSS('23,835689', '38,045108', '23,836809', '38,044776')),
POINTSST(
  POINTST('11.09.02 08:34:46', '11.09.02 08:35:16'),
  POINTSS('23,836809', '38,044776', '23,838958', '38,046374'))),
5)
```

Příloha B: Obsah přiloženého CD

Adresářová struktura přiloženého CD je následující:

- /*thesis* – text diplomové práce
- /*sources* – zdrojové kódy
- /*examples* – příklady použití
- /*data* – vzorová data
- /*documentation* – stručný návod